



Europäisches Patentamt  
European Patent Office  
Office européen des brevets



⑪ Publication number:

0 549 855 A2

⑫

## EUROPEAN PATENT APPLICATION

⑬ Application number: 92115676.6

⑮ Int. Cl. 5. A61F 2/64, A61F 2/68

⑯ Date of filing: 12.09.92

⑭ Priority: 05.12.91 US 804264

W-3408 Duderstadt 1(DE)

⑮ Date of publication of application:  
07.07.93 Bulletin 93/27

⑯ Inventor: James, Kelvin B.  
7955 - 98 Avenue  
Edmonton, Alberta T6A 0B5(CA)

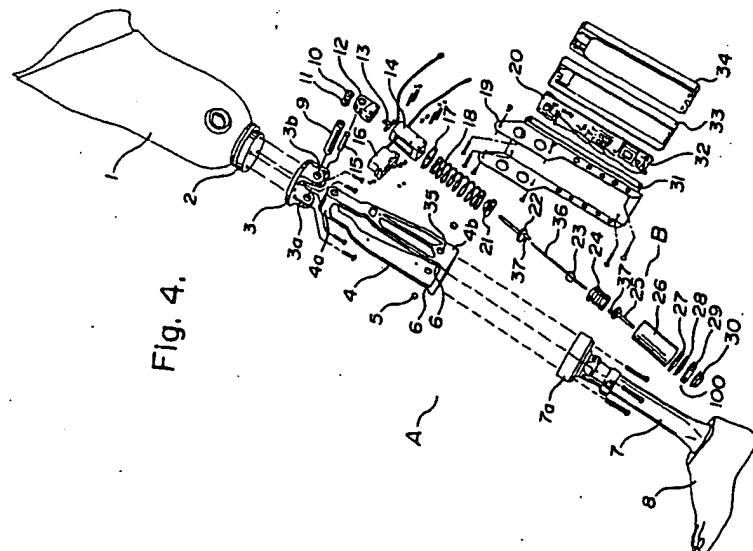
⑰ Designated Contracting States:  
AT BE CH DE DK ES FR GB GR IE IT LI LU NL  
PT SE

⑰ Representative: Gramm, Werner, Prof.  
Dipl.-Ing. et al  
Patentanwälte Gramm + Lins  
Theodor-Heuss-Strasse 1  
W-3300 Braunschweig (DE)

⑱ Applicant: Otto Bock Orthopädische Industrie  
Besitz- und  
Verwaltungs-Kommanditgesellschaft  
Industriestrasse

⑲ System for controlling artificial knee joint action in an above knee prosthesis.

⑳ This invention relates to an above knee prosthesis which employs a hydraulic damper to passively regulate the angular velocity or rotation of the artificial knee joint. A programmed microprocessor recognizes common gait patterns from information received from strain and knee angle sensors on the prosthesis. The microprocessor reacts at various transition points in the gait by activating a motor which in turn adjusts a valve assembly in the damper. The valve assembly is capable of variably and separately damping the knee joint motion in each of flexion and extension at the same time. Gait is improved because of the improved extent of control of knee action. In addition, distinct routines such as stair descending and sitting down can also be practised.



EP 0 549 855 A2

FIELD OF THE INVENTION

This invention provides a system for controlling the rotation of a knee joint of an above knee prosthesis. The system employs a microprocessor, responsive to lower leg strain and knee angle measurements originating from sensors on the prosthesis, to control a hydraulic damper through operation of a valve assembly associated with the damper, to thereby passively damp or resist the rotation of the artificial knee joint.

BACKGROUND OF THE INVENTION

As previously stated, the present invention is used with an artificial leg or prosthesis worn by an above knee amputee.

There are today about 50 different above knee prosthetic devices on the market. Many of these prostheses involve:

- 15 - a socket for receiving and engaging the stump of the user;
- a knee bracket rigidly connected to the socket;
- a frame extending down from the bracket and being pivotally connected to the bracket by a horizontal shaft, said bracket, shaft and frame together combining to form an artificial knee joint;
- a pylon and artificial foot connected to the base of the frame; and
- 20 - means for controlling the knee joint by locking it to prevent it from buckling under load in the stance phase of a step, and freeing it in the swing phase of the step.

Now, the biological or natural knee joint is powered by the actions of muscles. Muscle has two elements. One is the active force developed by contraction and the other is variable stiffness. It has not been feasible to duplicate muscle contraction in leg prosthetics, due to limitations arising from weight and bulk. As a result, research has focused on implementing stiffness into the knee joint. This has usually involved switching the knee joint between one of two modes: locked up or free to rotate.

In recent years, researchers have sought improvement in controlling the action of the artificial knee joint, as a way to improve gait and enable the amputee to better deal with certain distinct actions, such as descending stairs or lowering into a sitting position.

30 A relevant patent in this regard is French patent 2623-086-A. This patent teaches providing a strain gage sensor on the frame between the knee joint and foot, to measure load. The electronic signals from the sensor are transmitted to a microprocessor which monitors the load measurement. When the load signal indicates that the swing phase of the step is ending and load is being applied to the leg, the microprocessor causes a motor or electromagnet to lock up the knee joint. When the stance phase is complete, the microprocessor instructs the actuator to release the knee joint, so that it is free to pivot in the swing phase.

35 Another relevant prior art reference is Russian patent SU1333-333-A. This patent teaches using a sensor at the knee hinge, to measure knee angle. Means lock or free the knee hinge in response to the knee angle measurements.

40 Another relevant prior art device is known as the Henschke Mauch S-N-S system for controlling an above knee prosthesis. This system incorporates a linear hydraulic damper for resisting rotation of the knee joint at a single damping rate in the stance phase. The damping rate can be varied by manual adjustment. When the knee joint is fully extended, the damper assumes a non-resisting mode. Otherwise stated, the system lacks automatic variation of damping and incorporates only two states, namely high resistance to flexion in stance phase and free rotation in swing phase.

45 If a knee joint is looked at as a simple hinge, there are two separate actions which can occur. In "flexion", the knee joint rotates to enable the upper and lower leg segments to move closer together. In "extension" the knee joint rotates in the opposite direction, the leg segments move apart and the leg straightens. For an artificial knee joint to more closely simulate a biological knee joint, it is necessary that control or stiffness be applicable separately and variably in each of the flexion and extension modes. For 50 example, it is desirable at the beginning of the stance (i.e. weight bearing) phase of the step to allow a small amount of knee flexion to occur and to then lock the knee against further downward flexion while simultaneously freeing the knee to extend as the leg straightens due to body action. So in the latter phase of this action, the knee joint is altered to being locked or stiff in flexion and free in extension, at the same time.

55 To applicant's knowledge, there is no artificial knee joint mechanism disclosed in the prior art which enables separate, simultaneous and automatic variable control of flexion and extension.

If such a mechanism could be devised, then a much more sophisticated control over the knee joint action could be implemented.

It is the object of the present invention to supply such a mechanism and to then incorporate it in an improved overall prosthesis.

#### SUMMARY OF THE INVENTION

5 The present invention relates to an on-board, computer-directed system adapted to provide improved automatic control of knee joint rotation in an above knee prosthesis (AKP) having upper and lower leg segments joined by the knee joint, said lower leg segment having a foot. In general, the system comprises:

- 10 - a linear, hydraulic damper which can separately and variably damping or resisting each of flexion and extension rotational movements of the knee joint;
- electronic sensing means for measuring each of AKP knee angle and lower leg strain (which are respectively indicative of the angle between the leg segments and the position of the center of gravity of the user's body relative to the AKP foot) and emitting signals indicative thereof;
- actuating means, such as a servo motor, for adjusting the damping means to vary the resistance to 15 rotation of the knee joint in at least one of flexion and extension; and
- programmed computer means for receiving the emitted signals from the sensing means continuously establishing from said signals the state of the AKP in the course of a repetitive movement and activating the actuating means as required to vary damping to substantially simulate knee action. More particularly, the computer means is preferably adapted to do this by comparing the signals to 20 stored threshold values which are indicative of pre-determined transition point between states of the AKP in the course of a movement, and, when the received signal values correlate with stored values, then causing the actuating means to vary damper resistance as required so that the AKP knee joint action substantially simulates natural knee action.

It will be noted that the invention involves separate variation of damping of AKP knee joint action in 25 each of flexion and extension. "Damping" for this specification means resisting rotational movement of the knee joint. The resistance may be substantially complete, in which case the knee joint is substantially prevented from rotating in one or both of flexion and extension. The resistance may be partial, in which case the rate of rotation of the knee joint is restricted in one or both of flexion and extension. Or the resistance may be non-existent, in which case the knee joint is free to rotate in one or both of flexion and 30 extension. Alternatively stated, the damper is adapted to control the rate of rotation of the knee joint in one or both of flexion and extension.

To enable such bi-directional damping, applicant has developed a novel damper incorporating a piston and means for controlling the piston. More particularly, the variable, linear, hydraulic damper comprises:

- 35 - a hollow closed cylinder filled with hydraulic fluid and having a cylindrical hollow piston adapted to slide longitudinally within the cylinder chamber;
- the piston preferably has axial rods extending from its ends, which rods project through sealed openings in the end walls of the cylinder. The piston further carries an exterior circumferential seal ring between its ends, for sealing against the side wall of the cylinder;
- a first aperture and check valve assembly, associated with a first end wall of the piston, enables fluid 40 to enter the piston chamber from the first end of the cylinder chamber;
- a second aperture and check valve assembly, associated with the second end wall of the piston, allows fluid to enter the piston chamber from the second end of the cylinder chamber;
- a first pair of diametrically opposed ports extend through the piston side wall adjacent its first end, on one side of the seal ring;
- a second pair of diametrically opposed ports extend through the piston side wall adjacent its second 45 end, on the other side of the seal ring;
- preferably, each first port is offset circumferentially from the second port on that side of the piston;
- preferably, each port is slit-like in configuration;
- a valve preferably extends into the cylinder and piston chambers and is adapted to progressively 50 reduce or increase the effective area of the first (or flexion) ports available for fluid flow and separately progressively reduce or increase the area of the second (or extension) ports;
- most preferably the valve comprises a rotatable shaft extending into the piston chamber in parallel 55 relation to the cylinder axis, said shaft carrying a pair of radially protruding, diametrically opposed lobes, each lobe being adapted to substantially seal against the inside surface of the piston side wall, each lobe further being adapted, when the shaft is rotated, to progressively cover or uncover the adjacent flexion and extension ports, to thereby separately and simultaneously control flow area through the flexion and extension ports.

In use, one rod of the piston is connected to one segment of the AKP and the far end of the cylinder is connected to the other segment. For purposes of this description, it is assumed that the upper push rod of the damper piston is pivotally connected to the upper leg segment of the AKP and the lower end of the cylinder is pivotally connected to the lower leg segment. Therefore, in flexion the damper will contract and thus the piston will be driven downwardly in the cylinder by body load. In extension, the damper lengthens and the piston is pulled upwardly by body action.

- 5 In the operation of the damper:
- If the valve is positioned to enable flexion and if the piston is forced downwardly, thereby pressurizing fluid in the lower end of the cylinder chamber, fluid will flow upwardly through the lower check valve and extension ports, if open, into the piston chamber and will leave the piston chamber through the upper flexion ports - fluid will not leave the piston chamber through the extension ports (if uncovered) because there is no significant fluid pressure differential between the lower end of the cylinder chamber and the piston chamber;
  - If the valve is positioned to enable extension and if the piston is pulled upwardly, thereby pressurizing fluid in the upper end of the cylinder chamber, fluid will flow downwardly through the upper check valve and flexion ports, if open, into the piston chamber and will leave the piston chamber through the lower extension ports - again fluid will not leave the piston chamber through the flexion ports because there is no significant fluid pressure differential between the upper end of the cylinder chamber and the piston chamber.

- 10 20 It will be noted that the damper design is characterized by the following attributes:
- The valve can be adjusted to vary port areas and thus fluid flow rates to thereby vary resistance to knee joint rotation in either flexion or extension at the same time, thereby enabling variation of damping in both directions at the same time;
  - Because the ports are provided in diametrically opposed pairs, the valve does not get pressed against one side of the piston wall under heavy load and therefore does not seize up or become difficult to move - thus a small motor and shaft can be used to control the damper, which contributes to the compactness and lightness of the unit;
  - Because the damper is hydraulic, it is not significantly affected by wear and remains substantially consistent in its damping performance, thereby enabling the user to become accustomed to its "action" and to gain confidence in its performance. One could argue that the temperature of the hydraulic oil could vary and this would affect consistency of performance but this effect is minimized by using aircraft hydraulic fluid.

- 25 30 35 In a broad aspect, the damper design therefore involves providing:
- a pair of closed chambers (for example the two ends of the cylinder chamber);
  - means (for example the piston and cylinder) connected to the leg segments and forming two passageways (for example each formed by a check valve assembly, the piston chamber and a pair of the ports), for moving or pumping fluid from one end chamber to the other through one of the passageways when the leg segments are moving together and through the other of the passageways when the leg segments are moving apart; and
  - means (for example the valve and port assembly) for regulating the flow of fluid through each passageway.

40 45 In another aspect of the invention, advantage is taken of the repetitive nature of leg actions. If, for example, one is walking along a level surface, there are patterns of knee angle and lower leg strain measurements which do not change significantly from step to step. By monitoring the two sets of signals and timing, the computer software can determine the stage or state of AKP motion and can initiate appropriate changes in flexion and extension capability. If there is deviation from the regular pattern, such as stubbing the AKP toe in the course of swing phase, the software can detect this change and initiate corrective action.

50 Thus the system incorporates a method for controlling the knee joint of an AKP, which can be stated in the case of level walking as follows:

- storing, in a computer memory, threshold values of lower leg strain and knee angle, which values are indicative of the knee bending in stance phase, of anterior positioning of the center of gravity of body weight relative to the ankle or foot, and of swing phase, all in the course of a step along a level surface;
- continuously sensing lower leg strain and knee angle during use of the AKP and producing electronic signals corresponding thereto;
- comparing the signals against the stored threshold values and, when the signals substantially correlate with threshold values, actuating means for altering the rate of rotation of the knee joint in at least one

of flexion and extension to enable the knee joint to flex at about the beginning of stance phase, to lock the knee joint against flexion while enabling extension in the middle portion of stance phase, and to free the knee joint as it approaches the swing phase, thereby substantially simulating natural knee action; and

- 5 - repeating the foregoing repetitiously.

By combining the sensing means, the damper having means which can simultaneously and separately control flexion and extension and the software based on the profiles of repetitive motion, a knee joint system has been evolved which is characterized by closely controlled, predictable responses. This results in the user gaining confidence in the system which then manifests itself in the form of a longer and more rhythmic gait. The software can react similarly whether the gait is fast or slow. And the software can be "fine tuned" to the particular user to gain further compatibility or altered to modify the operation of the AKP. In addition, the system is adaptable to controlling the knee joint in the course of actions other than level walking, such as stair descent and sitting.

From the foregoing, it will be understood that the invention utilizes programmed computer means for receiving the emitted signals from the sensing means, continuously establishing from said signals the state of the AKP in the course of a movement and activating the actuating means to vary damping to substantially simulate natural knee action. More particularly, the programmed computer means is adapted to compare the emitted signals against stored threshold values indicative of transition points between states of a repetitive movement of the AKP and, when the signals substantially correlate with threshold values, to alter 20 the rate of rotation of the knee joint in one or both of flex-on and extension. Preferably, the stored threshold values are selected from the group consisting of the absolute and derivative values of knee angle and the position of the center of gravity of the user's body relative to the AKP foot, the duration from the last transition point and the possible future states in the course of the movement.

The invention described can be thought of as a machine which reacts to the amputee's movements, 25 thus improving gait. Confidence in the machine is necessary for the amputee to take full advantage of the machine's capabilities. This confidence is developed by ensuring that the machine reactions are reproducible, step after step.

In order to obtain consistent and reproducible reactions, the invention takes advantage of the reproducible mechanics of the prosthesis during normal walking. As previously stated, during each step the knee 30 goes through a pattern of movement which is basically the same, step after step. Also reproducible from step to step are the strains on the frame of the AKP, developed by the weight of the amputee, and the angle changes of the knee joint.

The repetitive nature of the signals is an important aspect of the success of the invention. This allows the prosthesis to have consistent man/machine interactions. The prosthesis is a tool used by the amputee to 35 perform different tasks. If the performance of this tool is predictable and reproducible, then user confidence is gained.

With the reactions occurring at the same time and in the same manner for each step, the amputee develops trust in the machine and is able to walk with a continuous fluid motion.

In summary, the invention works on the principle that each step can be divided into segments or states 40 and that a machine reaction can be developed for each segment, thus improving gait. The division of the step is carried out by first obtaining information from the prosthesis, conditioning this information with electronics and analyzing it with software, and then implementing machine reaction by separately and simultaneously varying resistance to flexion and extension rotation of the knee joint.

45 **DESCRIPTION OF THE DRAWINGS**

Figure 1 is a block diagram showing the flow of information in the system;

Figure 2 is a perspective simplified view of the Hall effect sensor used for providing signals indicative of knee angle;

50 Figure 3 is a plot of knee angle sensor output versus knee joint rotation;

Figure 4 is a perspective view of the prosthesis in exploded form;

Figure 4A is a perspective view of the prosthesis in assembled form;

Figure 5 is a plot of strain sensor output versus strain or load on the prosthesis;

Figure 6 is a diagram showing the states in level walking, with the appropriate state conditions shown;

55 Figure 6a is a diagram showing the states in level walking and correlating them with leg action, piston position and valve position;

Figure 7 is a plot showing the relationship between knee angle and strain (ankle bending moment or load) signals, related to the states, for level walking;

- 5 Figure 7a is a diagram showing the states in stair descent and correlating them with leg action, piston position and valve position;

10 Figure 8 is a diagram showing the states in sitting down, with the appropriate state conditions shown;

15 Figure 8a is a diagram showing the states in sitting down and correlating them with leg action, piston position and valve position;

20 Figure 9 is a plot showing the relationship between knee angle and strain signals, related to the states, for sitting down;

25 Figure 10 is a diagram showing the states in stair descent, with the appropriate state conditions shown;

30 Figure 11 is a plot showing the relationship between knee angle and strain signals, related to the states, for stair descent;

35 Figure 12 is a comprehensive diagram showing the states and conditions for the various modes of action;

40 Figure 12a is a comprehensive diagram corresponding with Figure 12 and showing the various body actions;

45 Figures 13 and 14 are simplified sectional side views showing the piston and cylinder in flexion and extension modes;

50 Figure 15 is a simplified end view of the internals of the piston;

55 Figures 16 - 24 are views similar to Figure 15, showing the valve in various positions;

60 Figure 25 is a side sectional view of the cylinder and piston;

65 Figure 26 is an overall circuit diagram of the system;

70 Figure 27 is a diagram of the communication circuit;

75 Figure 28 is a diagram of the microprocessor chip;

80 Figure 29 is a diagram of the voltage references and regulator for the analog to digital convertor located on the microprocessor chip;

85 Figure 30 is a diagram of the conditioning electronics for the Hall effect sensor;

90 Figure 31 is a diagram of the conditioning electronics for the strain sensor;

95 Figure 32 is a diagram of the conditioning electronics for low battery detection;

100 Figure 33 is a flow chart of the software and Figure 34 is an interrupt service routine which is activated every 20 milliseconds; and

105 Figure 35 is a perspective view showing strain gauge positioning on the base of the frame.

## DESCRIPTION OF THE PREFERRED EMBODIMENT

Having reference to Figures 4a and 4b, the prosthesis A comprises a suction socket 1 which is custom fabricated to closely fit the stump of the amputee and to cling to it by suction. An adjusting plate 2 is attached to the base of the socket 1. A knee bracket 3 is secured by screws to the adjusting plate 2. The knee bracket 3 has apertured shaft supports 3a, 3b for receiving, supporting and affixing the main knee joint shaft 9 and the damper shaft 15 respectively. A frame 4, having a bearing 4a at its upper end, is rotatively mounted to the knee bracket 3 by the main shaft 9, which extends through the bearing 4a. The frame 4 is therefore free to rotate or pivot on the fixed main shaft 9. At its lower end, the frame 4 forms a rectangular socket member 4b for receiving a rectangular block 7a which is clamped to the upper end of the foot pylon 7. Screws secure the pylon block 7a to the frame socket member 4b. A foot 8 is secured to the lower end of the pylon 7.

An upper bearing housing 12 is mounted for rotation on the damper shaft 15. The damper shaft 15 is located to the rear of the main knee joint shaft 9, so that the shaft 15 and upper bearing housing 12 follow an arc relative to the shaft 9 when the knee bracket 3 rotates or pivots.

A Hall effect sensor 13, shown in Figure 2, is provided to monitor the change in knee angle or knee joint rotation. The sensor 13 used is available from Sprague Electronics and is designated as model UGN-3503U. This sensor 13 comprises a ring magnet 11, which is fixed to the stationary damper shaft 15 of the knee bracket 3 by a ring magnet keeper 10. The sensor 13 further comprises a Hall effect transducer 13a, which is located in the rotatable upper bearing housing 12 and which is positioned facing the ring magnet 11. As knee joint rotation occurs, the bearing housing 12 moves around the damper shaft 15, causing the transducer 13a to move relative to the ring magnet 11.

The transducer 13a has a voltage output which is dependent on the magnet flux intensity (north or south pole) directly before it. Therefore, as the knee joint rotates, the output of transducer 13a changes. The signal from the linear Hall effect transducer is amplified to produce .5 volt with a knee joint extended fully and 4.5 volts with the knee joint flexed fully. Included in the circuit is a gain adjustment and an offset control. Stated otherwise, the signal of the transducer 13a is lowest when the knee is straight and increases

as the knee is bent. Figure 3 shows a typical sensor voltage output with respect to knee angle after amplification.

The forces on the foot 8 are established by measuring the strain of the frame 4. This is done using foil strain gauges 6 available from Micro Measurements Group Inc., Raleigh, North Carolina under designation 5 CEA - 06 - 062 UW-350. Four gauges 6 are used, two at the front and two at the rear of the frame 4, located between the frame apertures 101 and the base of the frame 4, to measure and differentiate between load on the heel and load on the toe of the foot 8. Stated otherwise, the strain measurement provides an indication as to whether the user body center of gravity is in the anterior, centred or posterior position relative to the AKP foot. The four gauges are wired in a wheatstone bridge configuration to produce an 10 electric signal which changes proportionally with strain. The wheatstone bridge configuration is a standard arrangement for determining the resistance change of strain gauges. The output of the bridge is amplified by a differential instrumentation amplifier 126 to produce an output signal of .5 volts when the heel is loaded fully and 4.5 volts when the toe is loaded fully. No load or similar load on the toe and heel produces 2.5 volts. Included in the circuit is gain adjustment and an offset adjustment. Figure 6 shows a typical voltage 15 output of the bridge with respect to foot loading after the signal is amplified. It will be noted that the load signal decreases as the heel is loaded and increases as the toe is loaded.

A servo motor bracket 14 is secured to the base of the bearing housing 12. A servo motor 16 is mounted within the bracket 14. The motor used is available from Airtronics Ltd. under designation 94737.

An upper spring retainer 17 is mounted on the base of the servo motor bracket 14, for a purpose to be 20 described.

A damper B is positioned between the servo motor bracket 14 and the base of the frame 4.

The damper B comprises a hollow cylinder 26, which is externally threaded. A lower spring mount ring 27 is threaded onto the outside surface of the cylinder 26, for a purpose explained below. A lower bearing mount ring 29 is also adjustably threaded onto the outside surface of the cylinder 26, at its lower end. The 25 ring 29 has radially extending threaded bores 100, normal to its central axis, which fit lower bearing pins 5 which are threaded through apertures 101 in the base of the frame 4. Thus the base of the cylinder 26 is pivotally coupled to the base of the frame 4 by threading the pins 5 into the bores 100 of the ring 29. A lock ring 28, threaded onto the external surface of the cylinder 26, is tightened against the ring 29 to lock it in place.

30 A lower cap 30 fits into the bore 102 of the cylinder 26 at its lower end and closes the bore. The lower cap 30 is held in place by a snap ring 103. The lower cap 30 carries a circumferential O-ring 104, for sealing against the side wall 105 of the cylinder 26. An aperture 106 is formed through the cap 30. An O-ring 107 is mounted in this aperture 106, sealing around the dummy push rod 25 of a piston 24.

At its upper end, the cylinder 26 has an upper cap 21 which fits into the cylinder bore 102 and is held 35 in place by a snap ring 108. The upper cap 21 also carries a circumferential O-ring 109, for sealing against the side wall 105 of the cylinder 26. An aperture 110 is formed through the cap 21. An O-ring 111 is mounted in this aperture 110, for sealing around the push rod 22 of the piston 24.

The hollow cylindrical piston 24 is positioned in the cylinder bore 102. The piston 24 comprises an open-ended drum 112 having upper and lower end caps 113, 114 screwed thereinto. A push rod 22 extends 40 upwardly from the upper end cap 113, through the sealed aperture 110 in the cylinder cap 21, and is secured to the servo motor housing 14. From the foregoing, it will be noted that the bearing housing 12, servo motor housing 14 and push rod 22 form a train of components connected to the damper shaft 15 and bracket plate 3. Thus as the socket 1 pivots about the main shaft 9, this rotational movement is converted into linear movement of the push rod 22 and piston 24.

45 A tubular spring 18 extends concentrically around the cylinder 26 between the upper spring retainer 17 and lower spring mount ring 27, for assisting the assembly to increase rate of knee extension during the swing phase of gait. This is useful in enabling increased speed of gait.

The piston 24 and cylinder 26 are shown in simplified form in Figures 13 and 14, with the fluid flows identified by arrows in each of flexion and extension.

50 The cylinder 26 is a closed or sealed unit and it is filled with hydraulic fluid. The piston 24 carries an external circumferential ring seal 115 for sealing against the side wall 105 of the cylinder 26.

The upper cap 113 of the piston 24 has an aperture 116 opening into the piston chamber 117. A spring-loaded one way check valve 118 controls the aperture 116 and allows pressurized hydraulic fluid to move downwardly from the upper end of the cylinder chamber 119 into the piston chamber 117.

55 The lower cap 114 of the piston 24 has an aperture 120 opening into the piston chamber 117. A spring-loaded one way check valve 121 controls the aperture 120 and allows pressurized fluid to move upwardly from the lower end of the cylinder chamber 119 into the piston chamber 117.

The check valves used are available from the Lee Company, Westbrook, Connecticut, under designation CKFA 2506205A.

5 A first pair of diametrically opposed flexion ports 122 extend through the piston side wall 123 at a point above the piston circumferential seal 115. A second pair of diametrically opposed extension ports 124 extend through the piston side wall 123 at a point below the circumferential seal 115.

From the foregoing and having reference to Figure 13, when body weight acts downwardly on the push rod 22 and piston 24, with the flexion ports 122 open, hydraulic fluid may flow upwardly from the lower end of the cylinder chamber 119, through the lower check valve 121 into the piston chamber 117, out of the piston chamber through the flexion ports 122 and into the upper end of the cylinder chamber 119. 10 Therefore, as long as the flexion ports 122 are open, the piston 24 may move downwardly, the damper B may contract and flexion of the knee joint may occur. If the flexion ports 122 are only partly open, there is damping or resistance to the knee rotation in flexion. If the flexion ports 122 are closed, the piston 24 is prevented from moving downwardly and the knee joint is locked against flexion.

Similarly, having reference to Figure 14, when the push rod 22 and piston 24 are pulled upwardly, with the extension ports 124 open, pressurized hydraulic fluid may flow downwardly from the upper end of the cylinder chamber 119, through the upper check valve 118 into the piston chamber 117, out of the piston chamber through the extension ports 124 and into the lower end of the cylinder chamber 119. Therefore, as long as the extension ports 124 are open, the piston 24 may move upwardly, the damper B may extend and extension of the knee joint may occur. If the extension ports 124 are only partly open, there is damping or resistance to knee extension. If the ports 124 are closed, the piston 24 is prevented from moving upwardly and the knee joint is substantially locked against extension.

As previously stated, restriction of the fluid flow through the ports reduces the flow of fluid through the hollow piston, thereby controlling the rate of movement of the piston.

The rate of flow of the fluid is controlled by an adjustable rotatable valve 23. This valve 23 is illustrated in Figures 4, 4b and 16 - 24. It comprises a shaft or rod 36 carrying a pair of lobes 125. The rod 36 extends axially and centrally into the piston chamber 117. It further extends upwardly through a bore 126 in the push rod 22 and is drivably connected with the servo motor 16 housed in the bracket 14.

The lobes 125 extend radially from the rod 36, substantially seal against the inside surface of the piston side wall 123 and each is adapted to extend vertically across both the upper flexion port 122 and the lower extension port 124 on one side of the piston 24.

The associated ports 122, 124 on each side of the piston 24 are circumferentially offset, as shown in Figures 16 - 24. Stated otherwise, the lower extension port 124 begins approximately where the upper flexion port 122 ends. The ports 122, 124 are narrow elongate horizontal slits. Typically they might have a length of .25 inches and width of .02 inches.

35 Therefore, there is a progressive nature to the reduction and subsequent increase in open area of a port as the valve lobe moves across it on a rotational travel. This of course affects the rate of fluid flow through the piston chamber 117 and determines the relative damping or resistance to rotation experienced by the knee joint.

By circumferentially offsetting the associated pair of upper and lower ports, there is a sequential and separate nature to the opening and closing of flexion and extension ports.

Stated otherwise, and as shown in Figures 16 - 24, the flexion and extension ports of an associated pair of ports on one side of the piston:

- 45 - can each be separately progressively opened or closed; or  
- each can be separately fully opened or closed; or  
- one can be fully closed while the other is progressively closed; or  
- both can be fully closed,

and all of the foregoing can be accomplished with a single motor and valve, thereby assisting in achieving compactness and low weight.

The rotation of the inner valve 23 is determined by the software controlling a microprocessor 32, which in turn controls the servo motor 16.

Each step or movement of the prosthesis has been divided into segments (states), dependent on comparison of the incoming sensor signals and preset threshold values. Held in the memory of the microprocessor is a position signal for the inner valve 23. With each change from state to state the inner valve 23 position is altered, thus achieving a different knee joint control. For example, referring to Figure 6A, 55 state No. 1, the initial portion of stance phase, the inner valve 23 is set to allow fluid to escape from the flexion ports 122 and consequently the knee joint can bend as the amputee applies weight. The programmed computer monitors the increasing knee angle and when it reaches the stored threshold value that indicates that the knee has bent to the predetermined angle initiating state No. 2, then the position of

the inner valve 23 is altered to completely restrict fluid flow from the flexion ports 122 and allow flow from the extension ports 124. This stops further knee joint bending and allows extension.

The above example illustrates that the assembly can have different control parameters depending on the direction of knee joint rotation (i.e. locked in flexion and allow extension). The fluid passes through separate ports for each of the two directions of knee movement. Therefore, if the flexion and extension ports are restricted independently of each other, the control of the rate of piston movement can be different for each direction.

The Figures 16 - 24 show discrete positions for the inner valve 23. In fact the positioning of the inner valve can be set at any position from 0 to 100 degrees, thus obtaining virtually an infinite range of knee joint damping. This is desirable for "tuning" the leg in activities such as stair descending, where the rate of descent must appeal to the amputee.

The microprocessor 32 used is available from Motorola Semiconductors Ltd. under designation XC 68 HC 811 E2 FN. This is an 8 bit processor having 2K of memory, 8 analog to digital convertors, and 8 digital inputs. The chip is about 1" x 1" and there is no need for any other peripheral chips, thereby allowing it to fit into a small package within the prosthesis A.

The knee angle and load sensor signals are amplified and then fed directly into the microprocessor 32. The amplifiers 126, 127 used for knee angle and load signal conditioning are available from Texas Instruments under designations TLC 272 and TLC 274 respectively.

As shown, the amplifiers 126, 127 and microprocessor 32 are mounted on a circuit board 20 and are enclosed together with a battery 34 (Motorola SNN 4038A) and battery holder 33 in a shell 19 which is secured to the frame 4.

## SOFTWARE

25 The software is set forth in the flow chart and attached Appendix.

Due to the similarities of the sensor information during the course of each step from one step to another (repetitiveness) it is possible to determine the amplitude of each of the two signals at transition points during each step. These transition points are important times when the damping of the knee joint should be altered to allow the amputee to walk. The transition points are detected by the processor 32 by comparing the predetermined "threshold" values, stored in memory, with the real signals from the prosthesis A and cycling through the transition points as they occur. As long as the amputee continues to produce signals as expected, the processor can keep track of the cycle.

With this type of software in operation the hydraulic damper B can be adjusted as each transition point occurs, to a new position which was predetermined during fitting.

35 This system can therefore determine the position of the prosthesis A during the course of each step and apply an appropriate damping coefficient to the knee joint. Furthermore it is possible to detect whether the amputee is walking on level ground, down stairs, sitting down or has encountered a dangerous situation such as the toe of the prosthesis hitting the ground during swing phase (toe stubbing).

## 40 Level Ground

Figure 6A illustrates the point. Each of the numbered circles are referred to as states. The processor always begins in state #1 where the step begins. As the amputee applies weight to the prosthesis A the knee joint begins to bend. This increases the knee angle signal which is continuously being compared to a preset threshold value and as it equals or exceeds the threshold value the processor cycles to state #2. The hydraulic damper setting is altered at the transition point to predetermined settings to allow knee flexion while in state #1 and to lock knee flexion while in state #2.

During state #1 the damper's function is to damp knee flexion and simultaneously allow knee extension and during state #2 to lock knee flexion and simultaneously allow but damp knee extension. Note that the flexion damping has gone from a damped setting to a locked setting independent of the damped knee extension setting. This design allows the amputee to straighten the knee during state #2 even though the knee flexion is still locked.

The damped setting is required to control the rate of knee extension as the amputee proceeds. If a free extension setting was chosen the knee would "snap" straight giving the amputee a noticeably abnormal gait.

The initial knee flexion after heel contact and the straightening of the knee is found in normal gait patterns and is referred to as "knee bounce".

The exact mechanics as to how the hydraulic damper functions is shown in Figure 6A beside each numbered circle.

Figures 6 and 7 show the rules used for the comparison and the actual values of the output of the sensors expected for one step. Following through the step it can be seen that the transition from state #1 to state #2 occurs as the knee angle signal in Figure 7 increases.

The graph shows that knee flexion stops shortly after the transition to state #2. The time delay is the time required for the damper to change.

As the amputee proceeds through the step the next important event is swing phase (time while the foot is off the ground). Indication of the oncoming swing phase can be detected by continuously monitoring the load signal and comparing it to a predetermined value.

As the centre of gravity of the amputee pass over the foot, weight is applied to the toe. The increase in the load signal causes the processor to switch to state #3 as soon as the load signal is equal to or exceeds the predetermined threshold value. The damper is commanded to unlock the knee joint, thus allowing the amputee to initiate swing phase when ready.

The entire swing phase is tracked by the processor. The transition to state #4 occurs when the knee signal increases past a preset threshold value as the knee joint flexes during the initial portion of swing phase.

After state #4 the strain or load signal is ignored and the processor monitors the first derivative of knee angle. The derivative is an indication of the speed and direction of the knee rotation. As the knee joint reaches the maximum flexion during swing the derivative becomes zero and detection of this produces a switch to state #5. Note that the same command for the damper is maintained throughout states #3-4-5, that is, free flexion and free extension which allows swing phase to be completed.

Completion of the swing phase is detected when the knee angle signal decreases past a preset threshold value to indicate that the knee joint has extended back to the straight position. The processor switches to state #1 and the entire process is repeated as long as the amputee continues to walk on level ground.

#### Emergency Swing Phase Recovery (Stubbing the Toe)

The normal repetitive pattern of knee angle and strain information causes the processor to cycle through state #'s 1-2-3-4-5-1 (see Figures 6 & 7). When the toe of the prosthesis has contacted an obstacle during the swing phase the pattern is different. The pattern is now 1-2-3-4-5-6-1. After state #5 the processor monitors the knee angle derivative information and switches to state #6 if the first derivative has become positive, indicating that the knee is no longer extending but is now flexing (i.e. the obstacle has interrupted the normal velocity of the knee extension). During state #6 the damper is instructed to lock the flexion of the knee joint.

Additional state changes exist for the level walking diagram. Circumduction is the completion of the swing phase without flexing the knee joint. This is done by swinging the limb sideways in an arc to clear the ground instead of flexing the knee. Without the flexion of the knee during the swing phase the processor would switch from state #'s 1-2-3 and stop. This problem is alleviated by measuring the time that the processor is in state #3 and if the knee has not been flexed in a predetermined amount of time the processor switches back to state #1 regardless of any inputs.

#### Sit Down Mode

During the daily events there are times when the amputee is sitting for an extended period of time. The knee joint of the prosthesis should be in an unlocked position for this time in order for the amputee to position the leg in any desired position. For instance he may wish to have it flexed to place the foot under a chair, or in a right angle position to sit upright, or in a partially flexed position for sitting in a car. The positioning is done by manipulating the prosthesis usually with the hands or the contralateral (other) foot.

Sitting is accomplished by training the amputee to perform a certain move to instruct the processor of the attempt to sit down. Figures 8A and 8 show the cycle of states for sitting down. Figure 9 shows the change in signals for a typical sit down motion. Initially the processor will be residing in state #1. The amputee leans backward which increases the load on the heel of the prosthesis and begins to flex the knee joint. The processor switches from state #1 to state #2 as the knee signal passes a preset threshold value (see state change on Figure 9).

The load on the heel decreases the load signal past a preset threshold value and the processor switches to state #7. As soon as the processor switches to state #7, a timer starts and measures the time

which the load is present on the heel. After 1/3 of a second the processor switches to state #8 which commands the damper to allow knee joint flexion. The amputee bears weight on the prosthesis and descends to the chair at a controlled rate. Measurement of time is again made and the processor switches to state #9 after 3/4 seconds. This commands the damper to be free in both flexion and extension of the knee joint, allowing the amputee to manipulate the leg to be comfortable in the seated position. The processor will remain in state #9 until the knee joint is extended to the straight position thus decreasing the knee angle signal past a threshold value at which the processor switches to state #1.

#### Stair Descending

10 The usual method for an amputee to descend stairs is to use only his good leg to lower his body weight down each stair until his prosthesis contacts the next stair. He then repeats the motion again using the good leg. The prosthesis is not used at all and the descent is "one stair at a time".

15 The second method is for the more agile amputee and consists of the normal "step over step" approach but doing so with the knee having uncontrolled descent as his weight flexes the knee (jack knifing).

The present invention incorporates a method of first detecting the fact that the amputee is about to descend a step and then offering a controlled rate of descent.

20 In order to initiate the descending of stairs, the processor must receive the appropriate signals from the user. This is done by placing the heel of the prosthesis on the edge of the stair and applying weight. Similar to level walking the first state change is from state #1 to state #2 as the knee begins to flex (see above). At this point the load signal decreases (heel loading) and the processor switches to state #7 and then to state #10 as the load reaches a preset threshold value (see Figures 10 & 11).

25 Note that the amount of weight placed on the heel by the user determines whether the processor stops at state #7 (detects "sit-down") or continues to state #10 (detects "stairs"). The user is trained to apply the appropriate weight to instruct the processor correctly.

30 A timer is started when the processor switches to state #10. As long as the user maintains the load for 1/2 second the processor will then switch to state #11. During state #11 the damper is commanded to damp the flexion of the knee joint and allow extension. This damping is similar to the hydraulic control unit on a door. The rate at which the door can swing is controlled by the hydraulic fluid within the cylinder. For the knee this damping is preset dependent on the wishes of the user. Some like to descend stairs at a slow rate while others prefer a fast descent.

35 At completion of each stair the user descends the next step on his contralateral (other) limb. During this time the processor is waiting for the knee joint to extend during the swing phase. The extension reduces the knee signal past a preset threshold value and the processor switches to state #12. The damper is commanded to lock flexion and allow extension. The user again places the heel on the next stair and repeats the sequence 7-10-11-12 for each step. Note that the processor does not return to state #1 after each step. This is due to the lack of a complete extension of the leg prior to the next step.

40 Once the flight of stairs has been completed, the knee joint is extended to the straight position and the processor switches to state #1 as the knee angle is reduced to a preset threshold value. The choice between stairs, sit down or level walking is now available.

45 Figure 12 shows all of the states grouped together. At the beginning of each step the software detects whether the amputee is proceeding on level ground (state #'s 1-2-3-4-5-1), has stubbed the toe during a step on level ground (1-2-3-4-5-6-1), is sitting down (1-2-7-8-9-1) or is descending stairs (1-2-7-10-11-12).

The amputee need not push any buttons or turn any levers to instruct the processor to change functions for different terrains. Detection is automatically done in real time dependent on the movements of the amputee.

50 Additional features of the state diagram include a battery life saver. If the amputee stops for more than 3 seconds in states 1, 2 or 9 the processor stops powering the control motor and goes to a shutdown state.

A low battery warning beeper signals the user that battery replacement is required. In the event that the battery is completely depleted the damper is commanded to damp flexion and free extension prior to complete loss of power. This allows the amputee to still bear weight on the leg without excessive knee flexion until a charged battery is placed in leg. As the flexion is damped the swing phase must be accomplished by circumduction during this time.

APPENDIX

39  
40  
41  
42  
43  
44  
45  
\*\*\*\*\* text formats \*\*\*\*\*  
;ALLCAPS: main entry point or CONSTANTS  
;lower: local labels  
;Mixed Ram variables  
;name, subr  
;NAME; main commands dispatched

10

15

20

25

30

35

40

45

50

55

## EP 0 549 855 A2

```

46 ; generally a blank line follows changes in flow, or logical thought,
47 ; or before entry points.
48 ;*****
49
5 50 ;External addresses in current knee controller 5/88
51 ;Using EVM board defaults to simplify debugging
52 ;MC8 has RAM 0-7fff, ROM 8000-ffff, DACs ff00-ffff
53 ; RAM $0000 - $00FF 256 bytes or $12 in E9 chip
54 ; REGS $1000 - $10FF $8000 in MC8 IF needed
55 ; EEPROM $B800 - $B7FF 512 bytes
56 ; EPROM $FO00 - $FFFF 4k 2732 external
57 ; EPROM $D000 - $FFFF 12k in 68HC711E9
58 ; EEPROM $F800 - $FFFF 2k in -A2/E2 cpus
59 ; DACS $FF00 - $FFFF only 8 active, addr wraps.
60 ;*****
61
62 0000 RS equ $0000 ;Ram starts here
63 00FF TOS equ RS + $FF ;Top Of Stack for -A1 (256 bytes)
64 1000 REG equ $1000 ;Registers sit here
65
66 ;9/02/90 to use EVM for development
67 ;can't load EVM EEPROM @ $B800 directly with S records so we're stuck here
68 0001 CHIPA2 equ 1 ;if using -A2/E2 for code
69
70 [01] ifndef CHIPA2 ;if not -A2/E2
71 EE equ $B600 ;EEPROM starts here in -A1/E9 version
72 EES equ $F800 ;Rules stored here for 1st time up.
73 ;Bootup copies to EE if difference in 1st 2 bytes detected (date).
74 ROM equ $F900 ; 2732 starts @ $0FO00, 2764 @ $0E000 -A2 @ F800
75 else ;in a 2k EEPROM device we have...
76
77 F800 EE equ $F800 ;EEPROM starts here in -A2 version in Single Chip mode
78 F900 ROM equ $F900 ;start code here in single chip mode
79
80 ;EE equ $E800 ;EEPROM set here in -A2 version in Expanded mode.
81 ;ROM equ $E900 ;start code here in Expanded mode
82
83 F800 EES equ EE ;both in same place in -A2's
84 [00] endif
85 ;*****
86 ;***** cpu def's. Uses REG value defined above or defaults to $1000 IFNDEF
87 ;
88 0000 INCLUDE d:\a6811\ruis\asm\6811reg.asm
89 INCLUDE \A6811\6811REG.ASM
90 .NJUST 0
91 [01] ifndef REG ;default if not defined elsewhere
92 REG equ $1000
93 endif
94 ;**** 6811 CPU I/O PORTS
95 1000 PORTA equ REG + $00 ;Port A & Timer Funcs
96 1002 PIOC equ REG + $02 ;Port dir/stbe control
97 1003 PORTC equ REG + $03 ;Port C I/O
98 1004 PORTB equ REG + $04 ;Port B Out
99 1005 PORTCL equ REG + $05 ;Port C Latched in
100 1007 DDRC equ REG + $07 ;Data Dir port C
101 1008 PORTD equ REG + $08 ;Port D

```

50

55

## EP 0 549 855 A2

102	1009	DDRD	equ REG + \$09	;Data DiR port D
103	100A	PORTE	equ REG + \$0A	;Port E for digital in on A/D
104				
105		;***** TIMER / COMPARE REGS		
106	100B	CFORC	equ REG + \$0B	;Compare force reg
107	100C	OC1M	equ REG + \$0C	;OC1 action Mask
108	100D	OC1D	equ REG + \$0D	;OC1 Data
109	100E	TCNT	equ REG + \$0E	;Timer Counter reg
110	1010	TIC1	equ REG + \$10	;Input Capture 1
111	1012	TIC2	equ REG + \$12	;IC2
112	1014	TIC3	equ REG + \$14	;IC3
10	1016	TOC1	equ REG + \$16	;Output Compare 1
113	1018	TOC2	equ REG + \$18	;OC2
114	101A	TOC3	equ REG + \$1A	;OC3
115	101C	TOC4	equ REG + \$1C	;OC4
116	101E	TOCS	equ REG + \$1E	;OC5
117	1020	TCTL1	equ REG + \$20	;Timer Control Reg 1
118	1021	TCTL2	equ REG + \$21	;TC reg 2
15	1022	TMSK1	equ REG + \$22	;Timer MaSK interrupt reg 1
120	1023	TFLG1	equ REG + \$23	;Timer FLaG int reg 1
121	1024	TMSK2	equ REG + \$24	;Timer MaSK int reg 2
122	1025	TFLG2	equ REG + \$25	;Timer FLaG int reg 2
123	1026	PACTL	equ REG + \$26	;Pulse Acc ConTroL reg
124	1027	PACNT	equ REG + \$27	;PA CouNT
20	126			
	127	;***** SPI PORT		
	128	SPCR	equ REG + \$28	;SPI control reg
	129	SPSR	equ REG + \$29	;SPI status
	130	SPDR	equ REG + \$2A	;SPI data
	131			
25	132	;***** SERIAL COMMUNICATION INTERFACE REGS		
	133	102B	BAUD	equ REG + \$2B ;baud rate register
	134	102C	SCCR1	equ REG + \$2C ;SCI control register 1
	135	102D	SCCR2	equ REG + \$2D ;SCI register 2
	136	102E	SCSR	equ REG + \$2E ;SCI status register
	137	102F	SCDR	equ REG + \$2F ;serial communications data register
	138			
30	139	;***** A/D REGS *****		
	140	1030	ADCTL	equ REG + \$30 ;A to D control register
	141	1031	ADR1	equ REG + \$31 ;A/D results
	142	1032	ADR2	equ REG + \$32
	143	1033	ADR3	equ REG + \$33
	144	1034	ADR4	equ REG + \$34
35	145			
	146	;***** CPU CONTROL REGISTERS		
	147	1035	BPROT	equ REG + \$35 ;Block protect in -E8 flavour
	148	1039	OPTION	equ REG + \$39 ;system configuration options
	149	103A	COPRST	equ REG + \$3A ;COP arm/reset
	150	103B	PPROG	equ REG + \$3B ;EEPROM Control
40	151	103C	HPRI0	equ REG + \$3C ;highest priority 1-bit interrupt and misc.
	152	103D	INIT	equ REG + \$3D ;Rom & I/O mapping
	153	103F	CONFIG	equ REG + \$3F ;Cop/Rom/EEprom enables & EE adr in -A2
	154			
	155			
	156	0000	_PORTA	equ \$00 ;Port A & Timer Funcs
	157	0002	_PIOC	equ \$02 ;port dir/stbe control
45	158	0003	_PORTC	equ \$03 ;Port C I/O

```

159 0004 _PORTB equ $04 ;Port B Out
160 0005 _PORTCL equ $05 ;Port C Latched in
161 0007 _DDRC equ $07 ;Data DiR port C
5 162 0008 _PORTD equ $08 ;Port D
163 0009 _DDRD equ $09 ;Data DiR port D
164 000A _PORTE equ $0A ;Port E for digital in on A/D
165
166 ;***** TIMER / COMPARE REGS
167 000B _CFORC equ $0B ;Compare force reg
10 168 000C _OC1M equ $0C ;OC1 action Mask
169 000D _OC1D equ $0D ;OC1 Data
170 000E _TCNT equ $0E ;Timer Counter reg
171 0010 _TIC1 equ $10 ;Input Capture 1
172 0012 _TIC2 equ $12 ;IC2
173 0014 _TIC3 equ $14 ;IC3
174 0016 _TOC1 equ $16 ;Output Compare 1
175 0018 _TOC2 equ $18 ;OC2
176 001A _TOC3 equ $1A ;OC3
177 001C _TOC4 equ $1C ;OC4
178 001E _TOCS equ $1E ;OCS
179 0020 _TCTL1 equ $20 ;Timer Control Reg 1
20 180 0021 _TCTL2 equ $21 ;TC reg 2
181 0022 _TMSK1 equ $22 ;Timer MaSK interrupt reg 1
182 0023 _TFLG1 equ $23 ;Timer FLaG int reg 1
183 0024 _TMSK2 equ $24 ;Timer MaSK int reg 2
184 0025 _TFLG2 equ $25 ;Timer FLaG int reg 2
185 0026 _PACTL equ $26 ;Pulse Acc ConTrol reg
186 0027 _PACNT equ $27 ;PA CouNT
25
187
188 ;***** SPI PORT
189 0028 _SPCR equ $28 ;SPI control reg
190 0029 _SPSR equ $29 ;SPI status
191 002A _SPDR equ $2A ;SPI data
192
30 193 ;***** SERIAL COMMUNICATION INTERFACE REGS
194 002B _BAUD equ $2B ;baud rate register
195 002C _SCCR1 equ $2C ;SCI control register 1
196 002D _SCCR2 equ $2D ;SCI register 2
197 002E _SCSR equ $2E ;SCI status register
198 002F _SCDR equ $2F ;serial communications data register
35 199
200 ;***** A/D REGS *****
201 0030 _ADCTL equ $30 ;A to D control register
202 0031 _ADR1 equ $31 ;A/D results
203 0032 _ADR2 equ $32
204 0033 _ADR3 equ $33
40 205 0034 _ADR4 equ $34
206
207 ;***** CPU CONTROL REGISTERS
208 0035 _BPROT equ $35 ;Block protect in -E8 flavour
209 0039 _OPTION equ $39 ;system configuration options
210 003A _COPRST equ $3A ;COP arm/reset
45 211 003B _PPROG equ $3B ;EEPROM Control
212 003C _HPRI0 equ $3C ;highest priority 1-bit interrupt and misc.
213 003D _INIT equ $3D ;Ram & I/O mapping
214 003F _CONFIG equ $3F ;Cop/Rom/EEprom enables & EE adr in -A2
215

```

## EP 0 549 855 A2

```

216      ;**** Some standard constants
217      0000      NUL    equ $00      ;Null termination for strings
218      0001      EOT    equ $01      ;End of Text
219      000A      LF     equ $0A      ;LF
5       220      000D      CR     equ $0D      ;CR
221      0011      XON    equ $11      ;^Q
222      0013      XOFF   equ $13      ;^S
223      001A      EOF    equ $1A      ;^Z
224      001B      ESC    equ $1B      ;Esc code
225      0018      CAN    equ $18      ;^X
226      0020      SPC    equ $20      ;space code
10
227      .LIST
228      0000      END    ;of definitions
230
231      ;*****+
232
15      233      0004      SDRATE equ $4      ;debug step rate
234
235      [01]      ifdef MHZ7      ;if 7.3728Mhz ...
236      ;**** Servo specific values
237      BRATE equ $12      ;$12 = 9600 $?? = 300 @7.3728Mhz /3/4
238      RTCRAT equ 36864      ;real time clock = 20mSec for 7.3728Mhz xtal
20
239      MDEG equ 22      ;Approx scale tic to time for 7.3728
240      [01]
241      0030      BRATE equ $30      ;define default com rate $30 = 9600B $34 = 600B @8Mhz
242      9C40      RTCRAT equ 40000      ;real time clock 20mSec
243      0014      MDEG equ 20      ;Approx scale tic to time
244      [00]
245
246      ;**** Terminal I/O Codes
247      002A      PROMPT equ '*'      ;prompt char
248      003E      GRS    equ '>'      ;
249      003C      LTS    equ '<'      ;
250      003D      EOS    equ '='      ;
30      251      000D      EOL    equ CR      ;cr on input
252
253      ;**** EEROM use in single chip mode -A2
254      ;0000 000F      Date, Bdrate... CHNTBL
255      ;0010 01FF      Rule tables (up to code lower limit), currently 00FF
256
35
257
258      ;**** RAM use
259      ;00 - 0F Copy of EE being adjusted. Must Save to make changes permanent
260      ;10 - 1E Sreadr, Curnrbf, Currul, Scnrl, Outadr, Frctr, Frctim,
261      ;20 -      EMPTY
262      ;3E      Beeper timer (don't move, dup defn in rules for separate assembly).
40
263      ;40 - 4F Main Flags & ram variables
264      ;50 - 5F Analog FIFO 24 bytes + derivatives (8)
265      ;70 - 7F Timer OCx reloads & phases (yet to implement)
266      ;80 - BF I/O buffers & vars ;INBUF_STA used by pbee for EESAVE ram routine
267
45
268      ;C0 - FF stack space (E0 + mini) !!Each int uses 9 bytes of stack!!
269
270      ;**** Rule work space ****
271      0000      org RS      ;workspace for tuning/adjust

```

50

55

EP 0 549 855 A2

```

272 0010      RSIZ    equ 16      ;this many to store in EE at a time
273 0000      Wrkrul  ds RSIZ      ;space for copy of whatever we're adjusting
274
275      ;*****
5  276 0010      org RS+$10      ;start of ram tables
277 0010      Srcadr  ds 2       ;source address of rule # (or data) in ram
278 0012      curmbf  equ $       ;fix assembler stupidity (DO NOT MOVE Curmbf. BRULxx fixed ref
)
10  279 0012      Curmbf  ds 4       ;where active mode bit fields stored (32 MAX rules for now)
280 0018      stime   equ $       ;stupid assembler
281 0018      Stime   DS 2       ;shutdown timer (battery) DO NOT MOVE Stime. BRULxx fixed ref
)
15  282 0018      Curmod  DS 1       ;currently active mode. Currl1 MUST follow
283 0019      Currl1  DS 1       ;currently active rule.
284 001A      Outedr  DS 2       ;address of motor value for fired rule (gtme)
285
15  286 001C      Frkrul  ds 1       ;Forced rule, MUST precede Frctim
287 001D      Frctim  ds 1       ;Rule timer, MUST follow Frkrul. (SEARCH) & inits STD X
288
289 001E      Inhrul  ds 1       ;Rule to inhibit for time specified. Inhtim MUST FOLLOW
290 001F      Inhtim   ds 1       ;Inhibit a rule timer. Inhrul MUST PRECEED
291 0020      Scnrl1  DS 1       ;scan rule counter (SEARCH) MUST follow Inhtim
292
20  293 0021      Oc1time DS 2       ;OC1 period. Basic sample rate (not yet used)
294
295 0023      Rtime   DS 2       ;Beeper timer register
296
297 0025      Offadr  DS 2       ;address offset for print outs
298
25  299      ;Reg    ds 2       ;REG address for fewer bytes of LDX #REG (9 refs) not used yet
)
300
301      ;***** Main vars in RAM *****
302 003D      org RS+$3D
303 1008      BPRTT   equ PORTD      ;beeper port
304 0020      BBIT    equ $20      ;bit in PORTD controlling beeper (D5 = 'SS')
305 003D      Btime   ds 2       ;beeper command bytes (chg+cmd) Ctime MUST follow
306 003F      Ctime   ds 1       ;count of beeps MUST Follow Btime
307
308 0040      Flag1   ds 1       ;Bit flags & constants. Flag2 - 4 MUST FOLLOW in RAM
309 0080      XFRFLG  equ $80      ;Xoff received
35  310 0040      EFLG    equ $40      ;input echo on
311      ;LFLG    equ $20      ;line feeds
312      ;HFLG    equ $10      ;hex input      ;not used since commented out in Hex-b
)
in
313
314      ;NFLG    equ $02      ;negative on input ;not used. commented out in he
40  x-bin
315      ;DBFLG  equ $01      ;debug print outs
316
317
318 0041      Flag2   ds 1       ;Bit flag & constants. MUST follow Flag1!!
319      ;RRFLG  equ $80      ;Rule in RAM being used, DON'T OVERWRITE unless clear!
45  !
320
321
322 0080      HLTFLG  equ $80      ;not needed since TIMINT ISR scans up to fired rule
                                ;before returning.
                                ;halt the rules (batt test)

```

50

55

EP 0 549 855 A2

323	0040	FRFLG	equ \$40	;just fired a rule, shortens scan so glitches & loops ;visible and allows FORCER a quick exit when done ;basically one rule fires per TIMINT.
324				
325				
326	0020	MFLG	equ \$20	;clock mode changes (must use FLAGS keyboard command)
5	327	FFLG	equ \$10	;we're Forcing a rule flag (forcer)
328				
329	0008	CFLG	equ \$08	;Change of rule #, bkgnd prints new # if SET & clrs bi
330	0004	PFLG	equ \$04	;Path found flag (scan)
331	0002	SFLG	equ \$02	;Scanning rules (timint) flag, prevents recursive ints
10	332	TFLG	equ \$01	;Tune flag. No state changes permitted if SET
333				
334				
335	0042	Flag3	ds 1	;Bit fields for rules to use. MUST PRECEED Flag4
336	0043	Flag4	ds 1	;Spare flags, cleared at power up MUST FOLLOW Flag3
337				
15	338	Msbb	DS 1	;Most significant byte of keyboard input values
339	0045	Lsbb	DS 1	;Lsb " " ". MUST FOLLOW Msbb
340				
341	0046	Lcnt	DS 1	;line count, MUST precede Lth
342	0047	Uth	DS 1	;line length for MEMDMP, MUST follow Lcnt
343	0048	Acnt	DS 1	;where print header function starts counting
20	344			
345	0049	Dtemp	DS 2	;Derive scratch space (addd). @@@@ use stack??
346				
347	004B	Adjadr	DS 2	;address of memory being adjusted (ADJ sets, TUNE uses)
348				
349	004D	Tmot	DS 1	;Temporary motor output value during tuning
25	350			
351	0050	org RS + \$50		
352	0010	DSIZ	equ 16	;index to start of newest data
353	0050	Fifo	ds DSIZ	;Past 2 blocks A/D data. Lowest =oldest AO, Highest =newest A7.
354	0080	Anidat	ds 16	;Current analog data & derivatives saved here ;@ could save bytes if derivatives only on 4 channels
355				
30	356			
357				;setup in advance of changes to TIMINT to handle 4 trains simultaneously
358	0070	org RS + \$70		
359		Mtime		;Motor time. Same as OC2
360	0070	Oc2time	DS 2	;OC2 PW current motor timer value (LD_MOTOR & TIMINT)
361	0072	Oc3time	DS 2	;OC3 PW
362	0074	Oc4time	DS 2	;OC4 PW
35	363	Oc5time	DS 2	;OC5 PW
364	0078	OC2ticp	DS 2	;OC2 tick/phase
365	007A	OC3ticp	DS 2	;OC3 tick/phase
366	007C	OC4ticp	DS 2	;OC4 tick/phase
367	007E	OC5ticp	DS 2	;OC5 tick/phase
40	368			
369	0080	RAMEND:	equ \$	;visible marker
370				.page

45

50

55

## EP 0 549 855 A2

```

371      ;**** I/O Buffer definitions
372      ;@@@@ this can be reduced to byte pointers & indexing since ram < 256.
373      ; redesign En-que/de-que to be dynamic sized and placed! NO INBEDED CONSTANTS!
374      ; But first we get the sucker to work.
5
375
376 0080      org RS + $80      ;I/O Vars start here
377 0080      BUFFS  equ $          ;Buffers start here
378 0020      OUTSIZ equ $20      ;Output size, 32 bytes or so
379
380 0080      OUT_BFSTA    ds OUTSIZ      ;C start of buffer address
381      ;OUT_BFSTA    equ BUFFS      ;C start of buffer address
10
382 009F      OUT_BFEND    equ OUT_BFSTA + OUTSIZ-1      ;C end of buffer address
383 001F      OUT_BFMAX    equ OUT_BFEND-OUT_BFSTA      ;C almost full circular buffer
384 0000      OUT_BFMIN    equ $00      ;C almost empty CB
385
386 00A0      IOSTR   equ $      ;Vars started here
15
387
388 00A0      Out_hdptr  DS 2      ;Out buffer head pointer
389 00A2      Out_tptr   DS 2      ;Out tail pointer. Points one behind valid data
390
391 00A4      Out_bffil  DS 1      ;Out buffer fill (In_bffil MUST follow. STD in SCINI)
20
392 00A5      In_bffil  DS 1      ;In buffer fill (MUST follow Out_bffil)
393
394 00A6      In_hdptr  DS 2      ;In buffer head pointer
395 00A8      In_tptr   DS 2      ;In buffer tail pointer
396
397 00AA      IOEND   equ $      ;ended here
398 0016      INSIZ   equ $20-(IOEND-IOSTR)      ;Input size = block size - vars used size
25
399      ;i.e. we steal bytes from input buffer since almost always empty.
400
401 00AA      IN_BFSTA    equ $      ;C start of buffer address
402 00BF      IN_BFEND    equ IN_BFSTA + INSIZ-1      ;C end of buffer address
403 0011      IN_BFMAX    equ IN_BFEND-IN_BFSTA-4      ;C almost full Circular Buffer
30
404 0000      IN_BFMIN    equ $00      ;C almost empty CB
405
406      .page

```

35

40

45

50

55

# EP 0 549 855 A2

```

407          ;*****
408          ;**** START OF EPROM ****
409          ;*****
410 00AA      relative ;back to normal to keep branches happy
5 411          ;*****
412          ;subr to update EE from rom if dates differ
413          ;*****
414  (01)      ifndef CHIPA2 ;if not 2k EE
415          org $E000 ;here for now
416
417          ERUDATE:
418          ldd #EE      ;destination address for full update
419          std Srcadr ;starts here
420          ldd EES      ;new ROM date
421          cpd EE      ;with current Date
422          beq rporet ;not different (DATES) so no need to change all
423
424  15        ldd #BRATE ;$F803 ;FROM ROM 1st time.$30 = E/13/1 > 9600 $34 = E/13/16
425          > 600 baud
426          staa BAUD ;fix it
427          clr $4000 ;clr EVB sci flip flop to enable com Tx
428          cli          ;allow int's so data can get out
429
430  20        ldd RUDATE ;do 256
431          ldd ree2nd ;next 256
432
433          sei          ;stop ints until timer/mode inits done
434          rts
435
436  25        ;**** subroutine called to update EE with EPROM rules
437          RUDATE: ldy #uvmsg ;inform
438          jsr PMSG ;user
439
440          ldy Srcadr ;current source in ram (EE write adr)
441          :: beq rporet ;nothing to save, AN ERROR
442          ldd #EE      ;where EE sits
443          bne rporet ;not EE, skip it!
444
445          ldy #EES ;rom table
446          ldd ree2nd ;256
447
448          rpcomp: ldd 0,y ;get current ram data
449          cmpa 0,x ;compare with previous saved EE
450          beq rbdec ;same so skip pgmng this byte
451
452          pshb          ;save counter
453          pshy          ;save current pointer
454          ;dbg...
455          pshe          ;save data
456          jsr HOUTC2 ;dbg
457
458          puls          ;get back
459          pshe          ;save again
460          jsr HOUTS ;dbg
461
462          puls          ;get data back

```

```

463      ;...dbg
464      lddab #316      ;byte erase first
465      bsr rpbe       ;do it
466
5       467      cmpla #3FF      ;did we ONLY NEED an ERASE?
468      beq ?rffdon    ;yes, skip pgmng
469
470      lddab #32 ;program
471      bar rpbe ;now
472
10      473      ?rffdon:
474      puly          ;recover em
475      pulb
476
477      rbdec:    inx          ;bump pointers ahead
478
15      479      decb          ;end count down
480      bne rpcomp    ;until zero
481
482      rpblet:   rts          ;done update
483
484      ;**** code compression subr
20      485      [02]      rpbe:    ifdef COPON
486                  jar copset    ;reset cop
487      [02]      else
488      [01]      endif
489
490      ldy #2200      ;2500 = 10 msec@ 4cy/day 3333 = 10msec@3/dex + 3/bns
491
25      492      ;B set to erase or pgm cmd, Y with delay value, and X pointing to address,
493      ;A with value to pgm
494
495      ;rpbe:
496      stab PPROG      ;set EELAT
497      stas 0,x ;write or erase @ x
498      inc PPROG      ;EEPROM up
30      499      ?wt10:    dey          ;count down
500      501      bne ?wt10    ;until done
501      502      clr PPROG    ;finished
502      503      rts          ;return to EE code
503
35      504      uvmsg:   db "Updating E","E" + $80
505
506
507      [00]      ;**** end of ERUDATE
508      endif
508      .page
40

```

45

50

55

EP 0 549 855 A2

```

509 F800          org EES ;Rules start here
510
511 F800          RULSTRT    equ EE ;working EE is here
512 ; INCLUDE D:\A6811\RULEASM\BRUL29.ASM ;set of rules
513 ; INCLUDE ASM\BRUL29A.ASM      ;new set of rules
5  514 F800          INCLUDE ASM\BRUL30.ASM      ;Kelly's name
515 ; 14:40 11/12/91
516 ; Base Rules BRUL30.ASM
517 ; Example of RULE30 definition based on 01/15/91 basem1 updated to 4/30/91
518 ; 24 rules possible in this configuration, 3 modes supported, all dynamic
519 ; Revision History
10 520 ; 1/28/91 added battery rule, HDR's
521 ; 2/4/91 MBIAS, BDRATE & SDRATE added in alterable EE
522 ; 2/19/91 Fixed error in Mode table RULE8 has no cond :. DON'T SCAN IT!
523 ; 2/22/91 Increased size to 24 address & added spare byte to existing rules
524 ; 2/28/91 Added demo digital rules
15 525 ; 3/12/91 Adjusted rule values to new defaults
526 ; 3/37/91 Add structure for Flags in rules
527 ; 4/04/91 Reverse order of Forced rules & timer to add Inhibit rules
528 ; 4/9-12/91 Work on battery rules & refinements
529 ; 4/30/91 KBJ Remove unused rules
530 ; 5/03&9/91 Rework battery rules
531 ; 5/22/91 Add NCCL for debug width
20 532 ; 11/12/91 Changed some values for four Tony Semps leg
533 ; 11/14/91 rules for separate sit and stairs

534
535          public DATE,NUMRUL,NUMMOD,BFCNT,MBIAS,MODE00,BDRATE,CHNLST
536          public SUBADR
537 ;everything else should be computable from tables as offsets
25 538
539 ;*** Configuration defines
540 [01]          ifndef RULSTRT ;if not defined, must be separate assembly
541 ;RULSTRT EQU $B600 ;likely address if separate assembly
542 RULSTRT EQU $F800 ;likely address if separate load
543 ;BRATE EQU $30 ;usual baud rate 9600
544 ;BRATE EQU $32 ;usual baud rate 2400
545 Btime equ $30 ;site here for now
546 ;EE equ $B600 ;usual if stand alone
547 EE equ $F800 ;usual if stand alone
548 EES equ RULSTRT ;defaults saved here
549 ; extern Stime,Curmbf ;fix errors
30 550 curmbf equ $12 ;force a fix for DUMB assembler
551 stime equ $16 ;force a fix for DUMB assembler
552
553 [00]          endif
554 0000 0 equ RULSTRT-EE ;difference for absolute references
555
40 556 ;REVERSE equ 1 ;define if channel order reversed
557 ;May just use CHNLST to change em.
558
559 0080 FIN EQU $80 ;Add to final rule # in possible routes list
560
561 0080 ; We need some definitions to aid later specification of Digital rules
45 562 0080 DIG EQU $80 ;Bit high in first byte of rule if digital bits present in rul
563 0080 B7 EQU $80 ;MSB
564 0040 B6 EQU $40

```

50

55

EP 0 549 855 A2

565	0020	B5	EQU \$20
568	0010	B4	EQU \$10
567	0008	B3	EQU \$08
568	0004	B2	EQU \$04
569	0002	B1	EQU \$02
5	570	B0	EQU \$01 ;LSB
		;Digital rule form is:	
		;DBBBBBBBB ;D = 1, B = Bits that matter = 1, 0 = don't care.	
		;XXXXXXXXX ;Logic state (Value) for selected bits to meet condition.	
		; these two bytes present only if digital conditions are used. Dig bit (D) high	
10	575	h ; in first byte of rule.	
576			
577	0000	ANL	EQU \$00 ;ANaLog. Dig bit low
578	0040	CHN	EQU \$40 ;CHaiN bit high for additional analog conditions
579	0020	GTV	EQU \$20 ;Greater Than Value
580	0010	LTV	EQU \$10 ;Less Than Value
15	581	EQV	EQU \$00 ;EQal Value (don't use generally. GT or LT safer since exact ;match not reliable on analog inputs.
582		NEV	EQU \$30 ;Not Equal Value
583	0030	DVT	EQU \$08 ;Derivative (time 20 Msec)
584	0008	NONE	EQU \$7F ;Special case, ANL+CHN+NEV+DVT+CH7 is DON'T CARE state ;!!! No value follows a NONE condition. !!!
585	007F		
586			
20	587		
588	[01]	IFDEF REVERSE ;if not reversed	
589			
590	0000	CHO	EQU 0 ;Analog input channel numbers
591	0001	CH1	EQU 1 ;makes redefinition easier
592	0002	CH2	EQU 2
25	593	CH3	EQU 3
594	0004	CH4	EQU 4
595	0005	CH5	EQU 5
596	0006	CH6	EQU 6
597	0007	CH7	EQU 7
598	0000	K0	EQU 0 ;Knee
30	599	LO	EQU 1 ;Load
600	0003	BATT	EQU 3 ;battery
601			
602	[01]	ELSE	;reversed
603			
35	604	CHO	EQU 7 ;Analog input channel numbers
605		CH1	EQU 6 ;makes redefinition easier
606		CH2	EQU 5
607		CH3	EQU 4
608		CH4	EQU 3
609		CH5	EQU 2
610		CH6	EQU 1
40	611	CH7	EQU 0
612		K0	equ 7 ;Knee
613		LO	equ 6 ;Load
614		TO	equ 6 ;Temperature Knee?
615		T1	equ 5 ;Temperature Load Cell?
616		BATT	equ 4 ;battery voltage
45	617		endif end conditional
618	[00]		
619			
620			

.page

```

621 ;***** Start of Rule table in EEPROM
622 ;Rules stay in EEPROM unless modified.
623 ;RULRAM bit field determines if a ram copy of a rule is used since ram
624 ;is very scarce in single chip mode.
5 625 ;Modified rules are saved to EE when a different rule is selected
626 ;for modification.

627
628
629 ;***** START OF DATA *****
630 F800 ORG RULSTRT
10 631 F800 11 14 91 DATE: DB $11,$14,$91 ;in HEX II

632
633 F803 30 BDRATE: DB BRATE ;Baud rate saved here to permit changes
634 F804 30 MBIAS: DB $30 ;Motor bias stored here. This is current 11/14/91 value
635 F805 40 MSAFE: DB $40 ;Safe motor value used by BATTERY to lockup cylinder
636 F806 0F BRUL: DB $0F ;Battery shutdown rule to fire after timeout
637 F807 31 BPVAL: DB $31 ;Beeper value for battery
15
638
639 ;NUMber of RULEs in table. This determines how high the search proceeds,
640 ;AND how many bytes are set aside for RULEADR's and RULTIM's.
641 ;space for RULADR's is exact = 2*NUMRUL
642 ;space for RULTIM is modulo 8, ie. 1-8 = 1B, 9-16 = 2B, 17-24 = 3B etc.
643 ;Ditto MODE00, etc.
20 644
645 F808 11 NUMRUL: DB $11 ;last rule # scanned. 24 rules max in this config.
646 ;Limit of scan.
647 F809 17 MAXRUL: DB 23 ;Maximum legal rule # 0-23
648 ;NUMber of MODEs. This makes a dynamic upload without relink possible.
649 ;since all addresses can be computed as offsets from RULSTRT. (but aren't yet)
25
650
651 F80A 03 NUMMOD: DB $03 ;currently 3 supported. Limited by EE space & rule size.
652 ;7 max.
653
654 F80B 03 BFCNT: DB $03 ;# bytes in bitfields since we could have more bit fields than
30
655 ;NUMRUL would indicate. Applies to MODEs, RULTIMs, RAMRUL
656
657 F80C 02 NCOL: db $2 ;# columns to print during debug
658 ;CHANnel LIST allows dynamic redefinition of input channels without having
659 ;to mess with rules. Allows us to fiddle between systems easily
35 660 F80D CHNLST:
661 ; DB $10,$32,$54,$76 ;forward sequence
662 ; DB $87,$45,$23,$01 ;reverse
663
664 ;*****
665 F810 ORG RULSTRT + $10 ;Keep position stable if addition bytes added
40 666 ;above (0 free)
667
668 ;RULEs with TIMe constraints. Bit field here saves space & speed
669 ;Binary field with MSB of lowest byte = RULE00, LSB = RULE07, etc.
670
671 F810 71 RULTIM: DB %01110001 ;00-07 :01,02,03,07 have timer
45 672 F811 E0 DB %111100000 ;08-0f :08,09,0A have timer
673 F812 00 DB %000000000 ;10-17
674
675 ;RULEs with FLoGa as conditions.

```

50

55

```

676 F813 00      RULFLG: DB %00000000 ;00-07 ; have flag words
677 F814 00      DB %00000000 ;08-0f ;
678 F815 00      DB %00000000 ;10-17
679 F816
5      680      ;Bit field of active rules for a given mode. Same DEF'N as RULTIM
681      ;For 24 rules, 3 bytes required.
682      MODE00: ;3F0      ;NORMAL OPERATION
683 F816 7F      DB %01111111 ;00-07 ;01,02,03,04,05,06,07 active conditions
684 F817 2C      DB %00101100 ;08-0f ;0A,0C,0D active conditions
685 F818 00      DB %00000000 ;10-17
10     686
687      MODE01: ;3F1
688 F819 00      DB %00000000 ;00-07
689 F81A 00      DB %00000000 ;08-0f
690 F81B 00      DB %00000000 ;10-17
691
15     692      MODE02: ;3F2
693 F81C 00      DB %00000000 ;00-07
694 F81D 00      DB %00000000 ;08-0f
695 F81E 00      DB %00000000 ;10-17
696
697      ;@ or use Flag bits to inhibit/enable rules OE & OF.
698
699      ; and so on for additional modes
700
701      ;NOTE!! A rule's STATE number is determined by its POSITION in THIS table,
702      ;NOT its name (eg RUL0A). This makes choosing alternate rules easy, just
703      ;put the alternates' address in the substituted position in table.
704
20     705 F81F      RULADR:
706 F81F F851      DW RULE00-0;00-07
707 F821 F855      DW RULE01-0
708 F823 F85F      DW RULE02-0
709 F825 F868      DW RULE03-0
710 F827 F872      DW RULE04-0
30     711 F829 F878      DW RULE05-0
712 F82B F87C      DW RULE06-0
713 F82D F880      DW RULE07-0
714
715 F82F F888      DW RULE08-0;08-0F
716 F831 F88D      DW RULE09-0
35     717 F833 F892      DW RULE0A-0
718 F835 F898      DW RULE0B-0
719 F837 F898      DW RULE0C-0
720 F839 F89F      DW RULE0D-0
721 F83B F8A4      DW RULE0E-0
722 F83D F8A4      DW RULE0F-0
40     723
724 F83F F8A4      DW RULE10-0;10-17 ETC.
725 F841 F8A7      DW RULE11-0
726 F843 F8AF      DW RULE12-0
727 F845 F887      DW RULE13-0
728 F847 F88F      DW RULE14-0
45     729 F849 F8C7      DW RULE15-0
730 F84B F8CF      DW RULE16-0
731 F84D F8D7      DW RULE17-0
732

```

EP 0 549 855 A2

```

733          ;List of subroutine addresses. Accessed by output values $F8-$FF.
734 F84F      SUBADR:
735 F84F F80F      DW SUB00-0
5      736      ; DW SUB01-0 ;etc
737          ;users responsibility to NOT call non-existent subroutines!
738
739          ;List of data tables for output. Accessed by output values $EO-$EF.
740          ;NOT implemented for Kelly's version
741 F851      DATAADR:
742      ; DW DAT00,DAT01 ;etc
10      743          ;none
744
745          ;*** Start of main part of rule table ****
746          ;Maximum number of rules is 128, named from $00 to $7F
747          ;RULE00 is fired at power up and restart.
748          ;it should not be in active mode list.
749
750          ;RULE00:
751      ; DB $00 + FIN    ;No previous rule (startup) + Final rule in list
752      ; DB NONE       ;No conditions
753      ; DB $FO        ;Invoke mode change to 0, but no output value to mot
15      or
754      ; DB $01        ;Rule to start with in new mode. This byte is unique
20      755          ;to mode rules ($FO-F7). It IS NOT TIME!
756
757          RULE00:      ;BATTERY LOW (special rule does not appear to fire)
758 F851 80      DB $00 + FIN    ;No previous routes
759 F852 13      DB ANL + LTV + BATT ;Analog, Less than, Battery
25      760 F853 90      BTVAL: DB $90      ;5.28 volts
761 F854 F8      DB $F8        ;Call subroutine
762
763          RULE01:      ;START
764 F855 05      DB $05      ;#5  ;possible routes
765 F856 06      DB $06      ;#6
766 F857 09      DB $09      ;#9
30      767 F858 0C      DB $0C      ;#C
768 F859 90      DB $10 + FIN  ;#10 + Final
769 F85A 10      DB ANL + LTV + KO ;Analog, Less than, Knee
770 F85B 09      DB $09      ;Threshold
771 F85C 48      DB $48      ;Output value
772 F85D 0F      DB $0F      ;Force rule F if...
35      773 F85E FA      DB $FA      ;Time limit of 250 * 20 msec = 5.0 sec
774
775          RULE02:      ;KNEE FLEXION
776 F85F 01      DB $01      ;#1  ;possible routes
777 F860 8F      DB $0F + FIN  ;#F + Final
778 F861 20      DB ANL + GTV + KO ;Analog, Greater than, Knee
40      779 F862 0A      DB $0A      ;Threshold
780 F863 57      DB $57      ;Output value
781 F864 0E      DB $0E      ;Force rule E if...
782 F865 FA      DB $FA      ;Time limit of 250 * 20 msec = 5.0 sec
783
784          RULE03:      ;TOE LOAD (KNEE EXTENSION)
45      785 F866 01      DB $01      ;#1  ;possible routes
786 F867 02      DB $02      ;#2
787 F868 0D      DB $0D      ;#D

```

50

55

EP 0 549 855 A2

```

788 F869 0E      DB $OE      ;#E
789 F86A 8F      DB $OF+FIN  ;#F + Final
790 F86B 50      DB ANL+CHN+LTV+KO  ;Chain, Analog, Less than, Knee
5  791 F86C 0F      DB $OF      ;Threshold
792 F86D 21      DB ANL+GTV+LO  ;Analog, Greater than, Load
793 F86E 9A      DB $9A      ;Threshold
794 F86F 32      DB $32      ;Output
795 F870 01      DB $01      ;Force rule 1 if...
796 F871 2A      DB $2A      ;Time limit of 42 * 20 msec = 0.84 sec
797

10 798      RULE04:      ;FLEXION
799 F872 83      DB $03+FIN  ;#3 + Final  ;possible route
800 F873 20      DB ANL+GTV+KO  ;Analog, Greater than, Knee
801 F874 18      DB $1B      ;Threshold
802 F875 32      DB $32      ;Output
803

15 804      RULE05:      ;TERMINAL FLEXION
805 F876 84      DB $04+FIN  ;#4 + Final ;possible route
806 F877 58      DB ANL+CHN+LTV+DVT+KO  ;Chain, Analog, Less than, Derivative, Knee
807 F878 FF      DB -$1      ;Threshold
808 F879 28      DB ANL+GTV+DVT+KO  ;Analog, Greater than, Derivative, Knee
809 F87A E2      DB $E2      ;Threshold
810 F87B 32      DB $32      ;Output
811

20 812      RULE06:      ;STUMBLE
813 F87C 85      DB $05+FIN  ;#5 + Final  ;possible route
814 F87D 28      DB ANL+GTV+DVT+KO  ;Analog, Greater than, Derivative, Knee
815 F87E 02      DB $02      ;Threshold
816 F87F 53      DB $53      ;Output
817

25 818      RULE07:      ;SMALL HEEL LOAD
819 F880 02      DB $02      ;#2 ;possible routes
820 F881 0C      DB $OC      ;#C
821 F882 8E      DB $0E+FIN  ;#E + Final
822 F883 11      DB ANL+LTV+LO  ;Analog Less than, Load
823 F884 3B      DB $3B      ;Threshold
30 824 F885 57      DB $57      ;Output
825 F886 08      DB $08      ;Force rule 8 if...
826 F887 32      DB $32      ;Time limit 50 * 20 msec = 1.0 sec
827

35 828      RULE08:      ;SIT DOWN (forced)
829 F888 80      DB $00+FIN  ;No previous routes
830 F889 7F      DB NONE     ;No conditions
831 F88A 4E      DB $4E      ;Output
832 F88B 09      DB $09      ;Force rule 9 if...
833 F88C 64      DB $64      ;Time limit of 100 * 20 msec = 2.0 sec
834

40 835      RULE09:      ;SEATED
836 F88D 80      DB $00+FIN  ;No previous routes
837 F88E 7F      DB NONE     ;No conditions
838 F88F 32      DB $32      ;Output
839 F890 10      DB $10      ;Force rule 10 if...
840 F891 FA      DB $FA      ;Time limit of 250 * 20 msec = 5.0 sec
841

45 842      RULE0A:      ;LARGE HEEL LOAD
843 F892 87      DB $07+FIN  ;#7 + Final  ;possible route
844 F893 11      DB ANL+LTV+LO  ;Analog, Less than, Load

```

EP 0 549 855 A2

```

845 F894 2A          DB $2A      ;Threshold
846 F895 57          DB $57      ;Output value
847 F896 0B          DB $0B      ;Force rule B if...
848 F897 32          DB $32      ;Time limit of 50 * 20 msec = 1.0 sec
5
849
850          RULE0B:      ;STAIR DESCENT
851 F898 80          DB $00+FIN  ;No previous routes
852 F899 7F          DB NONE    ;No conditions
853 F89A 49          DB $49      ;Output
854
10
855          RULE0C:      ;STAIR SWING
856 F89B 88          DB $0B+FIN  ;#B + Final  ;possible route
857 F89C 10          DB ANL+LTV+KO ;Analog, Less than, Knee
858 F89D 26          DB $26      ;Threshold
859 F89E 57          DB $57      ;Output
860
15
861          RULE0D:      ;FALSE HEEL LOAD
862 F89F 07          DB $07      ;#7  ;possible routes
863 F8A0 8A          DB $0A+FIN  ;#A + Final
864 F8A1 21          DB ANL+GTV+LO  ;Analog, Greater than, Load
865 F8A2 40          DB $40      ;Threshold
866 F8A3 00          DB $00      ;Output
20
867
868          RULE0E:      ;SHUT DOWN
869          RULE0F:      ;SHUT DOWN
870          RULE10:      ;SHUT DOWN
871 F8A4 80          DB $00+FIN  ;No previous
872 F8A5 7F          DB NONE    ;No conditions
873 F8A6 00          DB $00      ;No output
25
874
875 F8A7 FFFF FFFF FFFF          RULE11: Dw $FFFF,$FFFF,$FFFF,$FFFF  ;some blank space
     F8AD FFFF
876 F8AF FFFF FFFF FFFF          RULE12: Dw $FFFF,$FFFF,$FFFF,$FFFF  ;some blank space
     F8B5 FFFF
877 F8B7 FFFF FFFF FFFF          RULE13: Dw $FFFF,$FFFF,$FFFF,$FFFF  ;some blank space
     F8BD FFFF
878 F8BF FFFF FFFF FFFF          RULE14: Dw $FFFF,$FFFF,$FFFF,$FFFF  ;some blank space
     F8C5 FFFF
879 F8C7 FFFF FFFF FFFF          RULE15: Dw $FFFF,$FFFF,$FFFF,$FFFF  ;some blank space
     F8CD FFFF
880 F8CF FFFF FFFF FFFF          RULE16: Dw $FFFF,$FFFF,$FFFF,$FFFF  ;some blank space
     F8D5 FFFF
30
881 F8D7 FFFF FFFF FFFF          RULE17: Dw $FFFF,$FFFF,$FFFF,$FFFF  ;some blank space
     F8D9 FFFF
35
882
883          ;additional rules would require that RULNUM be raised, and additional
884          ;bit field bytes for RULTIM and MODExx be added & DW's for RULADR's
40
885          ;
886
887          ;Subroutines and D/A tables would go here if used.
888 F8DF          SUB00:
889 F8DF 7D 00 17          tst $time+1  ;running already?
890 F8E2 28 0E          bne nobt  ;yes, don't mess up
45
891
892 F8E4 15 12 40          bclr <(cumbbf),$40  ;take rule 1 out of scan
893 F8E7 4F          clra  ;no old beep command
894 F8E8 F8 F8 07          ldab BPVAL  ;the value to beep

```

50

55

EP 0 549 855 A2

```

895 F8EB 00 3D          std Btime . ;new beeper command
896 F8ED CC 17 70        ldd #6000  ;shutdown time out
897 F8FO 97 16          staa Stime ;make it run
898 F8F2 39             nobt: rts
899
900
901
902
903 F8F3             ;SUB01:
904
905 F8F3             ;DAT00:
906
907
908 1003      DIGIN  equ PORTC      ;for now
909 1004      DIGOUT equ PORTB
910
911
912 F900             ;*****org ROM ;Code starts here*****
913
914
915
916
917
918
919 0003      Esiz   equ 3      ;current element size for CMDDISP
920
921
922
923 F900 41             CMDTBL: ;^S & ^Q are handled by Rxint/Txint
924 F901 FD45
925 F903 44
926 F904 FF80
927 F906 45
928 F907 FD2E
929
930
931 F909 12             db SPC      ; Eat character
932 F90A FBE2
933 F90C 54
934 F90D FD9F
935
936
937 F90F 53             dw NULL
938 F910 FCAC
939 F912 4D
940 F913 FECE
941
942
943
944
945
946
947 F915 46
948 F916 FFB3
949
950
951 F918 80
      ;Main Command Jump Table. Consists of Cmd byte followed by address to jsr
      ;Searched sequentially so put most likely commands first in list
      ;01/11/91 Redefine elements & routines
      ;*****
      Esiz   equ 3      ;current element size for CMDDISP
      CMDTBL: ;^S & ^Q are handled by Rxint/Txint
      db SPC      ; Eat character
      dw NULL
      db 'A'      ;Adjust values in rule table
      dw ADJUST
      db 'D'      ;Debug. State & A/D values in hex
      dw DUMP
      db 'E'      ;enter values
      dw ENTER
      db 'P'      ;Print something (debug aid)
      dw PRULS
      db 'R'-$40  ;^Restart to power-up values
      dw BEGIN
      db 'T'      ;Tune motor values. Stops rules. Adjusts motor
      dw TUNE
      db 'R'      ;Recall parameter set #
      dw RECALL
      db 'S'      ;Save RAM to EE
      dw SAVE
      db 'M'      ;Memory dump @ address
      dw MEMDMP
      db ?'      ;Help. Display list
      dw HELP
      db 'F'      ;Free up mode change
      dw FREMOD
      db 'L'      ;Lock to a mode
      dw LOCMOD
      db 'F'      ;Flags toggle
      dw FLAGS
      db 'T'-$40  ;^Test
      dw TEST1
      db $80      ;end of table character MUST be $80

```

# EP 0 549 855 A2

```

952
953 ;***** real time table for real time commands
954 F919 RTTBL:
955 F919 3C db '<' ;dec var
956 F91A FEF8 dw DECVAR
957 F91C 3E db '>' ;inc var
958 F91D FEF7 dw INCVAR
959 F91F 18 db ESC ;escape loop
960 F920 FE5E dw ESCEXIT
961 ; db '' ;escape
962 ; dw ESCEXIT
963 ; db '' ;escape
964 ; dw ESCEXIT
965 F922 3D db '=' ;direct assignment
966 F923 FEFF dw SETVAR
967 F925 2B db '+' ;inc address
968 F926 FD80 dw INCADR
969 F928 2D db '-' ;dec address
970 F929 FD82 dw DECADR
971 F92B 3A db ':' ;set address
972 F92C FD73 dw SETADR
973 F92E 80 db $80 ;end of table
974
975 ;*****
976 ; RTDSPH. real time dispatch.
977 ; < > inc/dec value, + - inc/dec address being modified
978 ; Returns to one ABOVE caller if ESC, SPC, .. entered.
979 ; Echoes variable value for each step
980 ; falls into CMDDISP
981 ; X is callers address on return to MAIN
982 ;*****
983 F92F RTDSPH:
984 F92F CE F9 19 ldx #RTTBL ;Real Time command TaBLE
985 ;FALL INTO CMDDISP...
986
987 ;*****
988 ;CMDDISP. command dispatch from table of input chars -> addresses
989 ;X points to table head. A has char to check for match.
990 ; If $80 at end of table found, sets Carry, CMD not found.
991 ; X & B altered initially. Commands do whatever...
992 ;*****
993 F932 CMDDISP:
994 F932 C8 03 ldb #Esiz ;get element size for loop use
995 F934 20 01 bra cmdip ;to middle. Fewer branches inside loop is faster
996
997 F936 3A cmdnxt: abx ;add B to X to get to next table entry
998
999 ; may be able to use $80 as flag & bvs to find end
1000 F937 cmdip:
1001 F937 6D 00 tst 0,x ;end of table is $80
1002 F939 28 0A bni cmderr ;yes. Cmd not found
1003
1004 F93B A1 00 cmpl 0,x ;byte match?
1005 ; bvs cmderr
1006 F93D 26 F7 bne cmdnxt ;get next entry
1007
1008 ; ldb #1 ;preload for toggle functions

```

50

55

EP 0 549 855 A2

1009 F93F EE 01	Idx 1,x	;get address of found command (@X + 1) into X
1010 F941 3C	pshx	;save on stack
1011		
1012 F942 DE 4B	Idx Adjadr	;preload for smaller RTDSPH code size
5 1013		
1014	jmp 0,x	;and do it. Called procedure does rts to main loop
1015 F944 39	rts	;which is really a jump to table entry on stack!
1016		
1017 F945 0D	cmderr: sec	;set carry
1018 F946 39	rts	;return with error
10 1019		
1020	.page	

15

20

25

30

35

40

45

50

55

## EP 0 549 855 A2

```

1021      ;*****
1022      ;TIME INT. OC1 ints here to get work done
1023      ;*****
1024 F947      TIMINT:
5      1025 F947  CE 10 00      ldx #REG      ;Base
1026 F94A  1D 23 7F      bclr _TFLG1,x,$7F ;OC1 int flag cleared if high, R/M/Write
1027
1028      ;since we should be able to do this in under 20msec, we'll reenable ints & pre
      V
1029      ;      cli      ;allow others to int (SCI mostly for debugging)
1030
10  1031 F94D  1C 00 80      bset _PORTA,x,$80      ;dbg raise flag OC1 for timing
1032      ;      bar fsetx ;dbg raise flag, X ok.
1033
1034      ;*** FIFO function shuffles old data in ram to make room for new.
1035 F950  3C      pshx      ;save #reg
1036 F951  CE 00 50      ldx #Fifo ;start here
15  1037 F954  C8 08      ldeb #DSIZ/2      ;loop count
1038
1039 F958  1A EE 08      ?mvdata: ldy DSIZ/2,X      ;get 2 bytes from here &
1040 F959  1A EF 00      sty 0,X      ;stuff here (FIFO runs toward lower mem)
1041
1042 F95C  08      inx      ;move pointer up
20  1043 F95D  08      inx
1044
1045 F95E  5A      decb      ;count down
1046 F95F  26 F5      bne ?mvdata      ;until done
1047
1048      ;      ldx #REG      ;reload register base for speed/size
25  1049 F961  38      pulx      ;recover #REG
1050
1051      ;      bclr _PORTA,x,$80      ;dbg measurement
1052      ;      bar fcirx ;dbg clear flag, X ok
1053
1054      ;@ we may want ONE pass on 4-7 to minimize skew between channels
1055 F962  1C 30 04      bset _ADCTL,x,$4 ;change A/D to 4-7 repeating
30  1056      ;      ldeas #$14      ;change A/D to 4-7 ONCE
1057      ;      stae _ADCTL,X      ;now
1058
1059 F965  EC 31      ldd _ADR1,x      ;get ADR1,2 data. A/D must be done after 20msec...
1060 F967  DD 60      std Anidet      ;stuff 1,2
1061
35  1062 F969  EC 33      ldd _ADR3,x      ;get ADR3,4
1063 F96B  DD 62      std Anidet+2      ;stuff 3,4
1064
1065      ;since we should be able to do this in under 20msec, we'll reenable ints & pre
      V
1066      ;      cli      ;allow others to int (SCI mostly for debugging)
40  1067
1068 F96D  EC 16      ldd _TOC1,x      ;current time of int
1069 F96F  C3 9C 40      addd #RTCRAT      ;20msec ;@use Ram value?
1070 F972  ED 16      std _TOC1,x      ;update compare time
1071
1072      ;@ this code should be changed to use the OCx to raise the output, oc1 to clea
45  1073 F974  DC 70      ldd Mtime      ;motor width
1074 F976  27 13      beq nopol      ;zero

```

50

55

EP 0 549 855 A2

```

1075
1076 F978 E3 0E
1077 F97A C3 00 0A
1078 F97D ED 18
5 1079
1080 F97F 86 C0
1081 F981 A7 20
1082
1083 F983 86 40
1084 F985 A7 0B
1085
1086 F987 86 80
1087 F989 A7 20
1088
1089 F98B      nopl:
1090 F98B 0E      cli      ;can now allow ints here so PW doesn't jitter
1091
15 1092      ;?wtad0: brcir _ADCTL,x,$80,?wtad0 ;wait for A/D done (just in case)
      se)
1093
1094 F98C EC 31
1095 F98E DD 64
1096
20 1097 F990 EC 33
1098 F992 DD 66
1099
1100 F994 1D 30 04
1101      :      ldd _ADR1,x      ;get ADR1,2
1102      :      std Anidat+4      ;stuff 4,5
1103
25 1104      :      ldd _ADR3,x      ;get ADR3,4
1105      :      std Anidat+8      ;stuff 8,7
1106
1107      ;*** Beeper check
1108      ;@ use X if possible & reload later
1109
30 1110 F997 18 DE 16
1111 F99A 27 15
1112
1113 F99C 18 09
1114 F99E 18 DF 16
1115 F9A1 26 0E
35 1116
1117 F9A3 14 41 80
1118 F9A8 B6 F8 06
1119 F9A9 C8 01
1120 F9AB DD 1C
1121 F9AD C8 05
40 1122 F9AF D7 3E
1123
1124      :      clrb      ;need zero
1125      :      bra cbeep      ;and shut up beeper
1126      :      bra ?batok      ;skip onward
1127
45 1128      :      ldd MSAFE      ;safety value
1129      :      jsr LD_MOTOR      ;do it now
1130      :      bra ?batok      ;skip onward

```

EP 0 549 855 A2

```

1131
1132 F9B1    ?batst:
1133   :  ldd #A000    ;the battery channel
1134   :  cmpa BTVAL    ;the threshold in EE
1135   :  bhs ?batok    ;not too low yet
5
1136
1137   :  ldd #6000    ;2 minutes
1138   :  std Btime    ;setup time out
1139   :  ldd #BVAL    ;get command
1140   :  staa Btime + 1 ;startup beeper
1141
10
1142   ::  ldd BVAL    ;beep value & repeat
1143   ::  std Btime + 1 ;cmd & repeat count
1144
1145 F9B1    ?batok:
1146   :  ****
1147   :  **** Beep: handled here
1148   :  @ This can be improved (how?)
15
1149
1150 F9B1  D6 3E  ldd Btime + 1 ;get command byte
1151 F9B3  27 2E  beq sbeep   ;nothing happening, return
1152
1153 F9B5  D1 3D  cmpb Btime    ;compare with previous command
1154 F9B7  26 1E  bne ton    ;command changed. Start with on time, set counter
20
1155
1156   :  ?nchg:      ;no change, must still be timing
1157 F9B9  18 DE 23  ldy Rtime    ;get counter
1158 F9BC  18 09  dey        ;count down
1159 F9BE  18 DF 23  sty Rtime    ;update
1160 F9C1  26 20  bne sbeep   ;continue. time not up yet
1161
25
1162   :  brclr BPRT-REG,x,BBIT,ton  ;gnd if beep currently off, compute on
      time
1163 F9C3  1E 08 20 10  brset BPRT-REG,x,BBIT,ton  ;v+ if beep currently off, compute on
      time
1164
1165 F9C7    cbeep:
30
1166   :  bclr BPRT-REG,x,BBIT    ;clear beeper bit
1167 F9C7  1C 08 20  bset BPRT-REG,x,BBIT    ;clear beeper bit
1168
1169 F9CA  54  ?toff:  lsr  ;shift 'OFF' time to low nybble
1170 F9CB  54  lsr
1171 F9CC  54  lsr
1172 F9CD  54  lsr
35
1173 F9CE  28 03  bne offok  ;we do have an off time, else we're done.
1174
1175 F9D0  7F 00 3E  dirbt:  clr Btime + 1 ;clear command. Done for now. (fall through for size)
1176   :  ;Btime + 1 = zero prevents beep timing until cmd changes
1177 F9D3  17  offok:  tba  ;copy
1178 F9D4  3D  mul    ;time squared
40
1179   :  ldd #10 ;10 *.020 = .2 sec ticks
1180   :  mul    ;times low order in B
1181   :  std Rtime    ;update counter
1182   :  bra cbeep   ;clear beep
1183
1184   :  ;& we could do repeat count down here
45
1185   :  ;& ldd Ctime    ;get count

```

## EP 0 549 855 A2

```

1186          ;&    beq ?mul10    ;we don't have a count
1187
1188          ;&    deca          ;count out
1189          ;&    staa Ctime    ;update
1190          ;&    beq cirbt :timed out. stop update, else
5
1191
1192 F9D5 20 07          bra mul10    ;continue with common code
1193
1194 F9D7          ton:
1195          ; bset BPORTR-REG,x,BBIT    ;turn on beeper bit
10
1196 F9D7 1D 08 20          bcir BPORTR-REG,x,BBIT    ;turn on beeper bit
1197 F9DA D7 3D          stab Rtime    ;update command change byte
1198 F9DC C4 0F          andb #30F     ;mask to low order nibble
1199
1200 F9DE 86 0A          mul10: ldas #10 ;10 *.020=.2sec ticks
1201 F9E0 3D          mul          ;compute
1202 F9E1 D0 23          std Rtime    ;update/start counter
1203
1204          ebeep:          ;end of beep functions
1205
1206          .....*****
1207          ;**** rule timing. Do we need to check SFLG?
20
1208 F9E3 DC 1C          ldd Frctrl    ;get forcing info. rule # & time
1209 F9E5 5D          tstb          ;timer running?
1210 F9E8 27 08          beq ?nfrc          ;nope, no forced time active
1211
1212 F9E8 7A 00 1D          dec Frctim    ;count down timer in RAM
1213 F9E8 28 06          bne ?nfrc          ;not yet timed-out
1214
1215 F9ED 97 20          staa Scnrl    ;force this rule (A from LDD)
25
1216
1217          :    ldas #'T' ;dbg
1218          :    jsr dbg          ;T
1219
1220 F9EF 8D 50          bsr FORCER    ;make it active NOW
1221
30
1222 F9F1 20 08          bra endtim    ;skip to end. only one rule fired per Oc1 int
1223
1224 F9F3 86 83          ?nfrc: ldas #(HLTFLG + TFLG + SFLG) ;flags to check
1225 F9F5 95 41          bits Flag2    ;in here
1226 F9F7 28 08          bne norule    ;blocked by tune or scan or HALT (bott low)
1227
35
1228 F9F9 14 41 02          bset Flag2,SFLG    ;prevent recursive calls to scan
1229
1230          :    bcir Flag1,DBFLG,?dscn    ;dbg
1231          :    jsr CRLF ;dbg scan
1232          :    ldas #CR    ;dbg
1233          :    bsr dbg          ;cr
40
1234
1235 F9FC 8D 08          ?dscn: bsr SCAN    ;scan rules if not recursive interrupt
1236
1237 F9FE 15 41 02          endtim: bcir Flag2,SFLG    ;clear the scanning flag, or force deadlock
1238
1239 FA01          norule:
45
1240          :    bsr fcir    ;dbg timing
1241 FA01 CE 10 00          ldx #REG    ;recover base
1242 FA04 10 00 80          bcir _PORTA,x,$80    ;always end INT with OC1 cleared

```

EP 0 549 855 A2

```

1243 FA07 3B          rti          ;all done this interrupt (approx 1.5msec total)
1244
1245 ;*****
1246 ;turning measurement flag for debugging/refinement
5 1247 ;fcir:
1248 ;fset:
1249 ;    idx #REG      ;base
1250 ;fcirx:
1251 ;fsetx:
1252 ;    ldax _PORTA,X  ;get flag
1253 ;    eora #$80      ;flip OC1
10 1254 ;    staa _PORTA,X  ;stuff it back
1255
1256 ;    tst _PORTA,X  ;sign?
1257 ;    bpl fs          ;raise since down
1258
1259 ;    bclr _PORTA,x,$80    ;lower OC1 output
15 1260 ;    RTS
1261
1262 ;fset: idx #REG      ;base
1263 ;fsetx:
1264 ;fs:   bset _PORTA,x,$80    ;raise OC1 output
1265 FA08 39          RTS
20 1266 ;15 bytes vs 10
1267
1268 ;*****
1269 ;SCAN. Scans rules & generates outputs
1270 ;Modifies all registers
1271 ;1/29/90
25 1272 ;*****
1273 ;    db "SCN"      ;dbg
1274 FA09
1275 ;SCAN:
1276 FA09 7F 00 20      clr Scnrl      ;clr counter
1277 FA0C 15 41 40      bclr Flag2,FRFLG ;clear 'Fired a Rule' flag
30 1278
1279 ;    idx #MODE00      ;from sprom during debugging
1280 FA0F CE 00 12      idx #Curmbr    ;from ram table
1281
1282 FA12 A6 00      ?getbf: ldax 0,x ;get active rules bit field
1283 ;    beq ?nxtbf      ;skip if none at all. ;low probability, not enabled
1284
35 1285 FA14 48      ?shrbf: asla      ;bits into carry
1286 FA15 24 08      bcc ?nrule    ;this rule not active
1287
1288 FA17 3C      pshx      ;this rule active, stack 'em
1289 FA18 36      psha
1290 ;    ber fset ;dbg
40 1291
1292 FA19 8D 36      ber GETRUL    ;and check rule conditions
1293 ;    ber fcir ;dbg
1294
1295 FA1B 32      puia      ;recover 'em
1296 FA1C 38      pulx
45 1297
1298 FA1D 12 41 40 16  bset Flag2,FRFLG,nxt ;fired a rule. Only one per int so skip out
1299 ;next call to scan will clear FRFLG

```

50

55

## EP 0 549 855 A2

```

1300 FA21 4D          tsta      ;set CCs
1301
1302 FA22 27 05      ?nrule: beq ?nxtbf ;nothing left in reg so get next byte
1303
1304 FA24 7C 00 20      inc Scnrl
5      1305 FA27 20 EB      bra ?shrbf ;next scan rule # & check for active status
1306
1307 FA29 08      ?nxtbf: inx ;next bit field
1308
1309 FA2A D6 20      ldab Scnrl ;retrieve it
1310 FA2C C4 F8      andb #3F8 ;mask out partial counts (0-7)
1311 FA2E C8 08      addb #38 ;increment to next whole byte of BFs
1312 FA30 D7 20      stab Scnrl ;put it back
1313
1314 ;      tba ;dbg
1315 ;      jsr HOUT ;dbg
1316
1317 FA32 F1 F8 08      cmpb NUMRUL ;hit limit?
1318 FA35 23 DB      bls ?getbf ;nope, go check next BF byte
1319
1320 ;(doneul):
1321 ;      ldas #'s' ;dbg
1322 ;      jsr OUT_QC ;dbg
20      1323
1324 FA37 39      nxt: rts ;done this pass.
1325
1326 ;messages sit here for bar limits
1327 FA38 CD      mdemsg: db "M" + $80 ;mode error
1328 FA39 D2      remsg: db "R" + $80 ;error message here for bar
1329
25      1330 ;*** Subr to get EE address of rules (not ram)
1331 FA3A F1 F8 09      GETEDR: cmpb MAXRUL ;limit to present maximum rule
1332 FA3D 23 07      bls ?mok ;ok (unsigned, but since 7f max, needn't be)
1333
1334 FA3F CE FA 39      idx #remsg ;error
1335 ;      bar mdeout ;indirect jmp to PMSG
30      1336 FA42 BD FE 71      jsr PMSG ;tell 'em
1337
1338 ;      ldab MAXRUL ;limit it to??
1339 ;      ldab #0
1340 FA45 5F      clrb ;limit to power up rule since we have an error
1341
1342 FA46 CE F8 1F      ?mok: idx #RULADR ;table start in EE
1343 FA49 58      asib ;*2 for words
1344 FA4A 3A      abx ;index into address table
1345 FA4B EE 00      idx 0,x ;get address into x
1346 FA4D 39      rts ;done
1347
1348 FA4E 14 41 12      FORCER: bset Flag2,FFLG + SFLG ;set Forced & Scanning flags
40      1349
1350 ;      ldas #'f' ;dbg
1351 ;      bar dbg ;f
1352
1353 ;we don't want to stack things up if He goofs and gets an infinite loop of
45      1354 ;forced rules so that we jump around & around & around & ....
1355 ;so we fall into...
1356 ;      jmp GETRUL ;force the rule search.

```

EP 0 549 855 A2

```

1357 ;will return to caller after rule fires (we hope)
1358 ;since FRFLG set too, getrl returns quickly too.
1359 FA51
1360 ;** subr to get rule from table, check conditions, and fire output & timer
5 1361 FA51
1362 ;bar getscn :get the rule's address
1363 ;***** subr gets address of rule at Scnrl. Sits here for BSRs limits
1364 :gatscn:
1365 FA51 D6 20
1366 ;GETADR:
1367 FA53 8D E5
1368 FA55 9C 10
1369 FA57 28 03
1370
1371 FA59 CE 00 00
1372 ;sec ;NO, here is where THIS rule is
1373 ;?rak: ;and tell them so
1374
1375 FA5C 15 41 04
1376 ;?rak: bclr Flag2,PFLG ;clr path found flag
1377 ;dbg :brcr Flag1,DBFLG,getpath ;dbg printout
1378 ;dbg :jst HOUTC2 ;newline & address
20 1380 ;idae Scnrl ;get rule #
1381 ;jst HOUTS ;print it
1382 ;rts ;dbg
1383 ;bra getpath ;dbg
1384 ;db "GP" ;dbg
1385
25 1386 FA5F getpath:
1387 FA5F E8 00
1388 FA61 C4 7F
1389 FA63 27 04
1390
30 1391 FA65 D1 19
1392 FA67 28 03
1393
1394 FA69 14 41 04
1395
1396 FA6C 6D 00
1397 FA6E 08
35 1398 FA6F 2B 02
1399
1400 ;% inx ;point to next path
1401 FA71 20 EC
1402
1403 FA73 12 41 10 04
1404 FA77 13 41 04 BC
40 1405 FA7B ?fvpth:
1406 ;idae #'p'
1407 ;bar dbg ;p
1408
1409 ;% inx ;next up
45 1410 ;** Check FLAG Bit field & byte
1411 ;pshx ;save current pointer
1412 ;ldx #RULFLG ;FLAGS bit field

```

EP 0 549 855 A2

```

1413      ; jsr BFSCN      ;check flags bit fields for rule # in Scrnrl
1414
1415      ; pulx          ;recover pointer
1416      ; bcc ?dtst      ;no flags to check. Test for digital condition
5
1417      ;@@ NEED TO ADD a Force flag test here, but no room for code
1418      ;@ breset Flag2,FFLG,?fvflg      ;force valid flags if set
1419      ;@ also need to change to same style as digital rules. ACTIVE bits & VALUES
1420      ;@ could then use subroutine for both too.
1422
10
1423      ; ldaa 0,x      ;get the data byte
1424      ; anda Flag3      ;AND in current flags
1425      ; eora Flag3      ;flip sense of HOT bits
1426      ; bne nxt        ;condition not matched, RTS
1427
1428      ;?fvflg: inx      ;skip to next condition
1429 FA7B E8 00      ?dtst: ldab 0,x      ;next condition byte
1430 FA7D 2A 13      bpl anrl        ;Analog if MSB=0
1431
1432 FA7F      digitl:
1433      ; ldaa #'d'      ;dbg
1434      ; bsr dbg        ;d
1435
20
1436 FA7F F4 10 03      andb DIGIN      ;AND in external data with loaded mask
1437 FA82 E8 01      eorb 1,x      ;flip sense bits as spec'd
1438 FA84 E4 00      andb 0,x      ;mask them too for robustness
1439
1440 FA86 08      inx          ;point to analog next. skip to sense byte
1441 FA87 08      inx          ; and analog conditions
25
1442 FA88 12 41 10 04      breset Flag2,FFLG,aniget      ;force valid digital, check on analog
1443
1444 FA8C C4 7F      andb #$7F      ;mask out DIG bit
1445 FA8E 28 A7      bne nxt      ;Condition not matched, get next rul (just an RTS)
1446
1447 FA90 E8 00      aniget: ldab 0,x      ;get analog conditions
30
1448
1449 FA92      anrl:
1450      ; ldaa #'a'      ;dbg
1451      ; bsr dbg        ;a
1452
35
1453 FA92 C1 7F      cmpb #NONE      ;check for don't care
1454 FA94 27 4C      beq fires      ;don't care analog cond, so special fire
1455      ;no threshhold follows this condition!
1456 FA96 12 41 10 3F      breset Flag2,FFLG,nxtcnd      ;force analog
1457
1458 FA9A C4 0F      andb #$0F      ;mask to channel bits only
1459 FA9C 3C      pshx          ;save pointer
40
1460      ;*****      ; here we put CHNTBL translator
1461      ;[01]          ifdef TRANSLATE      ;translator enabled
1462      ; tba           ;save for bit tests
1463      ; andb #$07      ;just A/D ch # bits
1464      ; lsrb          ;into nybbles
1465
45
1466      ; idx #CHNTBL      ;translate table
1467      ; sbx           ;index into table
1468
1469      ; ldab 0,x      ;new value

```

50

55

EP 0 549 855 A2

```

1470          bitb #$1 ;odd/even nybble?
1471          beq ?chevn      ;even ;change to bne to swap nybbles around
1472
1473          lsrb           ;shift into low nybble, 0 fill (NOT ASR!!)
5
1474          lsrb
1475          lsrb
1476          lsrb
1477
1478          ?chevn: bitb #DVT      ;38      ;derivative?
1479          beq ?chok      ;no, use as is
1480
10
1481          orab #DVT      ;raise this bit, (offset CH address by 8)
1482          ?chok: andb #$30F    ;keep within bounds (kill any high nybble residue)
1483          ;30 bytes if Y, 27, if X
1484          [00]          endif
1485          ;***** end of translator
1486
15
1487  FA9D C5 08          bitb #$8 ;DVT?
1488  FA9F 27 03          beq ?nodvt    ;nope
1489          ;      bra ?nodvt    ;dbg
1490
1491  FAA1 BD FB 92          jar DERIVE    ;set derivative for THIS channel into ram table
1492
20
1493  FAA4 CE 00 60          ?nodvt: idx #Anidat  ;analog data table
1494  FAA7 3A                abx          ;add in offset to channel
1495  FAA8 E6 00          lddb 0,x  ;get data value
1496
1497  FAAA 38                pulx         ;recover pointer
1498
25
1499          ;      brcr 0,x,DVT,?nobug    ;dbg only DVT instructions
1500          ;      pshx          ;dbg save pointer
1501          ;      lddc 1,x  ;dbg get threshold
1502          ;      xgdx          ;dbg swap to X
1503          ;      jar HOUTC2    ;dbg and print
1504          ;      xgdx          ;recover B
1505          ;      pulx         ;dbg recover pointer
30
1506          ;?nobug:          ;dbg
1507
1508  FAAB E1 01          cmpb 1,x    ;compare with threshold and branch as required
1509
1510
35
1511          ;@ we might try a data table, indexed by 3 bit offset
1512          ;currently branch costs 5 * 4 bytes = 20 vs 8 * 2 addresses + index overhead
1513          ; or
1514          ;?nodvt:
1515          ;1      pshb          ;save channels
1516          ;2      ldd #3          ;offset per branch
1517          ;1      mul
40
1518          ;3      addd $7b00    ;base
1519          ;2      xgdy          ;to Y
1520          ;1      pulb          ;recover channels
1521
1522          ;      idx #Anidat  ;analog data table
1523          ;      abx          ;add in offset to channel
45
1524          ;      lddc 0,x  ;get data value
1525
1526          ;      pulx         ;recover pointer

```

50

55

EP 0 549 855 A2

```

1527      :      cmps 1,x      ;compare with threshold and branch as required
1528      :
1529      ;3      jmp 0,Y      ;check them
1530      :
1531      ;= 13 + 6 for 2 more beq/bne, rts for dvt, may not be faster either.
5
1532
1533 FAAD 1E 00 30 19      brset 0,x,$30,?b110      ;this way NEV. MUST test BOTH bits FIRST
1534
1535 FAB1 1E 00 18 18      brset 0,x,$18,?b011      ;this way LTV DVT
1536 FAB5 1E 00 10 0B      brset 0,x,$10,?b010      ;this way LTV = BLO
1537 FAB9 1E 00 28 13      brset 0,x,$28,?b101      ;this way GTV DVT
1538 FABD 1E 00 20 06      brset 0,x,$20,?b100      ;this way GTV = BHI
10
1539      ;      brclr 0,x,$30,?b000      ;this way EQV, falls through.
1540
1541      ;We use hard coded values rather than defined constants
1542      ;since code sequence dependencies exist,
1543      ;but the following branch chain sorts it all out
1544 FAC1 27 18      ?b000: beq nxtcnd      ;condition ok, check for more
1545 FAC3 39      rts      ;condition NOT MET, try nextrul
1546
1547 FAC4 25 13      ?b010: bie nxtcnd
1548 FAC8 39      rts
1549
1550 FAC7 22 10      ?b100: bhi nxtcnd
1551 FAC9 39      rts
1552
1553 FACA 26 00      ?b110: bne nxtcnd
1554 FACC 39      rts
1555
1556 FACD 2D 0A      ?b011: bit nxtcnd      ;signed for DVT
1557 FACF 39      rts
20
1558
1559 FADO 2E 07      ?b101: bgt nxtcnd      ;signed for DVT
1560 FAD2      nomc:
1561 FAD2 39      nodbg: rts
1562
1563      ;this is here to allow bar's for debugging
30
1564      ;dbg: brclr Flag1,DBFLG,rtodbg      ;no debug
1565      ;      jmp OUT_QC      ;echo chars & return
1566
1567      ;sits here for bar's
1568 FAD3 CE FA 38      mth:    idx #mderrmsg      ;@ERROR ERROR1 mode too high for table
1569 FAD6 7E FE 71      mdeout: jmp PMSG      ;so print out, and abort change
35
1570
1571
1572      ;*****
1573 FAD9 1F 00 40 04      nxtcnd: brclr 0,x,CHN,fires      ;chain? = $40 No, fire analog if we got this far
1574
1575 FADD 08      inx      ;yes, point to next set (cond 1)
1576 FADE 08      inx      ;(and 2)
1577 FADF 20 AF      brcr aniger      ;and check it
1578
1579 FAE1 08      fires: inx      ;point to output byte (last cond)
1580
1581      ;special case, no conditions
40
1582 FAE2 08      fires: inx      ;(output)
45

```

EP 0 549 855 A2

```

1583
1584      :  Idee #1' ;dbg
1585      :  bsr dbg      ;ignite
1586
5       1587      ::  Idee Scnrl  ;we want to be here
1588  FAE3  DC 1F      ldd Inhtim  ;we want to be here (B = Scnrl follows Inhtim)
1589      ::  cmpb Currl  ;we are currently here
1590      ::  beq ?nfir  ;same as last time so skip firing calls
1591
1592  FAE5  4D      tsta      ;Inhtim is? (A of LDD Inhtim)
1593  FAE6  27 05      beq ?kfire  ;no inhibits
10      1594
1595  FAE8  D1 1E      cmpb Inhrul  ;does our scan rule match the inhibited rule?(B = Scnrl
1596      :  bne ?kfire  ;nope
1597  FAEA  20 01      bra ?kfire  ;dbg we don't care
1598
15      1599  FAEC  39      ?nfir:  rts  ;we didn't REALLY fire the rule. Look at next one
1600  FAED
1601      :  cmpb #1  ;special rule?
1602  FAED  5D      tstb      ;special rule? 11/12/91
1603  FAEE  27 0B      beq nosee  ;yes, don't let anyone (except SPI) see it fire
1604
20      1605  FAFO  D7 19      stab Currl  ;so tell everyone of new rule number
1606  FAF2  7F 00 1D      clr Frctim  ;new rule, clear timer.
1607      :  clr Inhtim  ;and inhibits? If inh & other fires, do we stay inh'd?
1608
1609  FAF5  14 41 48      beq Flag2,CFLG + FRFLG  ;tell bkgd & scan of rule number change
1610  FAF8  15 41 10      bcl Flag2,FFLG  ;and clear forcing flag
25      ;@ there is a logical inconsistency here. Fflg cannot be cleared here.
1611
1612
1613
1614
1615
1616
30      ;*** write rule # to D/A port via SPI
1617  FAFB  B6 10 29      nosee:  Idee    SPSR      ;Knock down any SPI flags
1618  FAFE  B6 10 2A      Idee    SPDR      ;and dump any read data
1619
1620      :  Ideb Currl  ;get the rule that fired (stab Currl above)
1621  FB01  58      asib      ;*2, 128 max (scale up output)
1622  FB02  58      asib      ;*4, 64 max
35      1623  FB03  58      asib      ;*8, 32 max
1624  FB04  F7 10 2A      stab SPDR  ; write to SPI port.
1625
1626      :  the      ;dbg
1627      :  jsr HOUTS  ;dbg
1628
40      1629  FB07  DF 1A      stx Outadr  ;save address of output value for timer
1630
1631  FB09  EC 00      ldd 0,x  ;get output value (A) & byte following (B)
1632  FB08  81 FC      cmpa #$FC  ;lower than?
1633  FB0D  25 17      bne notspl  ;not special
1634
45      1635  FB0F  27 0B      beq flagset  ;flagset function ($FC)
1636
1637  FB11  81 FE      cmpa #$FE  ;Beep?

```

EP 0 549 855 A2

```

1638 FB13 22 36          bhi digou .      ;Digital1 ($FF)
1639
1640 FB15 25 09          blo flagcr ;flagcr function ($FD)
1641                               ;(equal ($FE))
1642 FB17 4F      :BEEPER    ?beep:  clra
1643 FB18 DD 30          std Btime      ;startup beep timer with data byte B & no old command
5   1644 FB1A 20 32          bra dcont    ;and continue with common code
1645
1646 FB1C DA 42          flagst:  orab Flag3  ;"OR" in Flag bits
1647 FB1E 20 02          bra fcont   ;and continue with common code
1648
10  1649 FB20 94 42          flagcr:  anda Flag3  ; "AND" in Flag bits
1650 FB22 D7 42          stab Flag3  ;stuff 'em back
1651 FB24 20 28          bra dcont    ;and continue with common code
1652
1653 FB26      notapi:
1654 FB26 81 F0          cmppa #$FO  ;special?
1655 FB28 25 37          blo domot   ;nope, standard motor
1656
1657 FB2A 81 F8          cmppa #$F8  ;sub? with what's left
1658 FB2C 24 25          bhs dosuba  ;yes
1659
1660      ;fall into domode  ;$FO-$F7
1661      ;      bset Flag2,FFLG,domode  ;forcing a rule?
20  1662 FB2E 12 41 20 A0          bset Flag2,MFLG,nomc  ;no mode change allowed
1663
1664 FB32      domode:
1665      ;      ldaa #'m'
1666      ;      ber dbg   ;m
1667      ;      ;change mode table, called by LOCMOD also (maybe)
1668 FB32 80 F0          suba #$FO  ;subtract base
1669 FB34 B1 F8 0A          cmpe NUMMOD  ;limit is?
1670 FB37 24 9A          bhs mth   ;something wrong, over table so don't change yet
1671
1672 FB39 97 18          sta Curmod  ;update mode prompt
1673      ;ldd...  ldaa 1,x  ;get 1st rule in new mode
1674 FB3B D7 20          stab Scnrl  ;1st rule in new mode will be forced later
30  1675
1676 FB3D F8 F8 0B          ldd BFCNT  ;get bit field count
1677 FB40 3D              mul      ;mode # * size/mode in bytes
1678
1679 FB41 C3 F8 18          addd #MODE00  ;add in base of 1st table in EE
1680 FB44 8F              xgdx    ;swap to X
35  1681
1682      ;Not super fast, but smaller?? then a LDD 0,y, STD 0,x twice? NOPE 11 vs 8
1683      ;it IS universal for size changes though.
1684      ;      ldy #Curmbf  ;2 dest, active rules in ram
1685      ;      ldd BFCNT   ;3 # to xfer
1686
40  1687      ;we don't have to worry about ints here since SFLG should stop recursion
1688      ;      jsr COPYM   ;3 copy to ram (domode)
1689 FB45 8D FC 78          jsr copybf  ;save redundant loads by jumping into COPYR
1690 FB48 7E FA 4E          jmp FORCER  ;3 make it fire
1691
1692      ;      ldd 0,x    ;2 get data (8)
45  1693      ;      STD Curmbf ;2 put data
1694      ;      ldd 1,x    ;2 2nd word

```

## EP 0 549 855 A2

```

1695      :: STD Curmbf + 2 ;2 put
1696
1697
1698      ;***** ;Digital output only possible in single chip mode (supposedly)
1699      ;although we could use unused port A bits
5       1700 FB48      digou:
1701          ldaa 1,x ;get output pattern
1702          stab DIGOUT    ;output pattern to defined port
1703          inc Outadr + 1 ;point to real output for gtimer
1704          ;@@ bug if rollover occurs in low address byte if > 256 EE for rules
1705 FB51 20 10      bra gtimer ;digo
10      1706
1707      ;***** ;call a subroutine & setup timer
1708      dosubs: 1709 FB53 18      tsb      ;move value to useful place
1710          ldaa #';s'
1711          bsr dbg      ;s
1712      1713 FB54 C0 F8      subb #3F8    ;subtract base
1714 FB56 58      asib      ;*2 for addresses
1715 FB57
1716 FB57 CE F8 4F      idx #SUBADR ;get the base
1717 FB5A 3A      abx      ;index in
1718 FB5B EE 00      idx 0,x    ;get the address into X
20      1719 FB5D AD 00      jar 0,x    ;call it
1720
1721 FB5F 20 02      bra gtimer ;standard finish
1722
1723 FB61      domot:
1724 FB61 8D 13      bsr LD_MOTOR ;stuff timer
1725      ; then fall into gtimer
1726 FB63      gtimer:
1727 FB63 CE F8 10      idx #RULTIM ;timer bit fields
1728 FB66 8D 1A      bsr BFSCN  ;check timer bitfields
1729
30      1730 FB68 24 0B      bcc ?timdn ;no timer for this rule.
1731          ;C unchanged with DECA in BFSCN
1732 FB6A DE 1A      idx Outadr ;recover pointer to output value
1733 FB6C EC 01      ldd 1,x    ;get rule # & timer
1734 FB6E 2A 03      bpl ?settm ;positive rule #... forced timer
1735
1736      ;@? and a #$7F ;mask off high bit
35      1737 FB70 DD 1E      std Inhrul ;neg rule#, inhibit timer
1738 FB72 39      rts
1739 FB73      ?settm:
1740 FB73 DD 1C      std Frerul ;save here for timint
1741
1742      ; ldaa #';dbg
1743      ; jmp dbg      ;t
1744
40      1745 FB75 39      ?timdn: rts      ;done for now
1746          ;.page

```

45

50

55

EP 0 549 855 A2

```

1747 ;*****
1748 ; loc_motor: lock motor in safe position
1749 ;*****
1750 ; loc_motor: ;used by EEUDATE
1751 ; lddaa MSAFE ;get safe position from EE
1752 ; fell into LD_MOTOR
1753 ;*****
1754 ; LD_MOTOR. Scale motor value in !IA1 to 16 bit timer value Mtime
1755 ;*****
1756 FB76 LD_MOTOR:
1757 : cmpe Lstmot ;check previous setting (Last Motor Time)
1758 : beq ldr ;skip computes if same (for speed)
1759 FB76 4D
1760 : staa Lstmot ;save motor time for next time in
1761 FB76 27 03 tsta ;set CCs
1762 FB77 88 FB 04 beq zero ;no bias since zero requested
1763 FB79 04 adda MBIAS ;add in bias from EE
1764 ;@@@@ should be EE? so dynamic scaling possible
1765 zero: lddab #MDEG ;degrees -> time
1766 FB7C C8 14 mul ;compute it
1767 FB7E 3D std Mtime ;and stuff here for output ints
1768 FB7F DD 70
1769 FB81 39 ldr: rts
1770 .page

1771 ;*****
1772 ; BFSCN. checks a bit field for a 1 at the rule # in B
1773 ; and returns Carry Set if high
1774 ; X points to BF, uses Scrnrl for rule to check
1775 ; BFSCNB, B contains rule # being checked
1776 ; Alters A
1777 FB82 BFSCN:
1778 : psrb ;save
1779 FB82 D8 20 lddab Scrnrl ;rule being checked
1780 ;BFSCNB:
1781 FB84 17 tba ;copy
1782 FB85 54 lsr b ;into byte offset, no sign extension
1783 FB86 54 lsr b
1784 FB87 54 lsr b
1785
1786 FB88 3A abx ;index into bit field list
1787 FB89 E8 00 lddab 0,x ;get bits
1788 FB8B 84 07 anda #17 ;mask A to bit # in byte
1789
1790 FB8D 58 shib: lslb ;into carry
1791 FB8E 4A deca ;count down, C unaffected
1792 FB8F 2C FC bge shib ;until done 8 times
1793
1794 FB91 39 ; pulb ;recover
1795 FB91 39 rts ;return with Carry set/clr from chosen bit
1796 .page

```

EP 0 549 855 A2

```

1797      ;*** DVT code
1798      ;FIFO(16),A/D(8),DVT(8). Fifo runs toward lower addresses.
1799      ;0 $8 $10 $18
1800  FB92  DERIVE:
5       1801      ; bar fcir ;dbg flag
1802
1803  FB92  37      pshb      ;save B
1804  FB93  C6 00 50      ldx #Fifo ;base
1805  FB96  C4 07      andb #37      ;mask to channels only
1806  FB98  3A      abx       ;offset to channel #
10      1807
1808  FB99  A6 08      ldax $8,x    ;get f(1) =previous
1809  FB9B  C6 04      ldab #4
1810  FB9D  3D      mul       ;4*f(1)
1811  FB9E  DD 49      std Dtemp    ;save for later use
1812
1813  FBA0  A6 10      ldax $10,x   ;f(0)
1814  FBA2  C6 03      ldab #3
1815  FBA4  3D      mul       ;3*f(0)
1816
1817  FBA5  93 49      subd Dtemp    ;3*f(0)-4*f(1)
1818  FBA7  DD 49      std Dtemp    ;save again
1819
20      1820  FBA9  E6 00      ldab $0,x    ;f(2)
1821  FBAB  4F      clra      ;high order clear
1822  FBAC  D3 49      addd Dtemp    ;3*f(0)-4*f(1)+f(2)
1823
1824      ;by way of explanation for the following code
1825      ;DEC1 HEX
25      1826      ;385 0181 after ASR shift values
1827      ;129 0081
1828      ;128 0080 if msbyte is zero before shift then data will be ok, else pmax
1829
1830      ;001 0001
1831      ;000 0000
30      1832      ;01 1fff
1833      ;-127 ff81
1834      ;-128 ff80
1835      ;-129 ff7f if msbyte is FF before shift then data will be ok, else nmax
1836      ;-385 fe7f
1837  FBAE  4D      tsta      ;check high order info
35      1838  FBAF  27 0E      beq ?bok    ;plus normal, exit ok
1839
1840  FBB1  2B 04      bni ?ftst ;check negative
1841
1842  FBB3  C6 7F      ldab #127     ;positive limit
1843  FBB5  20 0A      bra ?sdvt
40      1844
1845  FBB7  81 FF      ?ftsc:  cmpe #$FF     ;other limit
1846  FBB9  27 04      beq ?bok    ;neg normal, exit ok
1847
1848  FBBB  C6 80      ldab #128     ;neg limit
1849  FBBD  20 02      bra ?sdvt ;store it
45      1850
1851  FBBF  47      ?bok:   asra      ;/2
1852  FBC0  56      rorb      ;the long way since no ASRD instruction
1853
50
1854  FBC1  E7 18      ?sdvt:  std $18,x    ;store here. DVT data above FIFO & A/D data
1855  FBC3  33      pulb      ;recover channels
1856      ; bar fcst ;clear debug flag
1857  FBC4
55      1858  FBC4  39      rts       ;all done
1859      ;18 bytes^
1860
1861      .page

```

```

1862      ;**** Sign-on message
1863      ;SIGNON:      db CR,LF,"03/28/91",,*+$80      ;say hello
1864
1865
1866
5 1867      ;***** MAIN PROGRAM *****
1868      ;***** certain OPTION, INIT & TMSK2 bits can only be written once
1869      ;up to 64 E cycles after reset, so we MUST configure
1870      ;here immediately after power-up or COP reset.
1871      ;once only at power up
1872  FBC5  CE 10 00      ldd #REG      ;register base
10 1873      ;  ldd #901      ;Ram @ $0, Reg @ $1000
1874      ;  staa $103D      ;INIT @ reset. But RESET put it this way ANYWAY
1875
1876  FBC8  88 92      ldd #992      ;for options...
1877      ;ADPU = 1 => on CSEL = 0 = E IRQE = 0 = Lvl DLY = 1 = D CME = 0 = off x = 0 CR = 11 = cop = .26 Sec
1878  FBCC  A7 39      staa _OPTION,X      ;make any changes now!
1879
1880  FBCC  8F 35      ldd #900      ;Block Protect off for now
1881      ;  staa BPROT      ;
1882      ;  clr _BPROT,X
1883
1884  FBCE  8F 24      ldd #900      ;timer base rate .5uSec @ 2Mhz. all ints off
1885      ;  staa TMSK2      ;set rate now!
1886
20 1887      ;This really should be done by EVM board since a change to CONFIG requires
1888      ;that ALL EEPROM in -A1 be erased (Bulk is only way to erase config)
1889      ;High 4 bits determine EEPROM location in -A2 chip (2k on upper 1/2 of 4k bound
1890      ;
1891      ; ifi expanded, else $F800 if single chip mode.
1892      ; EE overlays ROM in -A2 Single chip therefore no Buffalo routines access!
1893      ;  ldd #5ED      ; $E for EE in -A2, nsec = 1 nocop = 1 romon = 0 eon = 1
1894      ;  staa CONFIG      ;set config. But since EE cell don't have to.
1895
1896  FBDD  4F      ifdef ILLBUG      ;doing debug on illegal instructions traps
1897      ;usually cause by bad rule structures
1898  FBDD  5F      ldd #0000
30 1899  FBD1  5F      clra
1900  FBD2  DD 44      clrb      ;the small way to get D = 0
1901  FBD4  20 0C      std Msbb      ;prevent trash first time around
1902
1903      ;we end up here on an illegal instruction or SWI trap
35 1904      ;START: tx      ;get old stack for illegal instruction trap
1905  FBD6  30      stx Msbb      ;save for output later
1906  FBD7  DF 44      ldy #IN_BUFSTA      ;dest
1907  FBD9  18 C0 00 AA      ldd #18      ;this many
1908  FBDD  C8 10      jsr COPYM      ;copied (start) & PRAY stack is OK!?
1909  FBDF  BD FC 7D      ;endif
40 1910
1911      ;*****
1912      ;normal restart of program (^R)
1913      ;*****
1914  FBE2  BEGIN:
1915  FBE2  0F      sei      ;mask off int's if on
1916  FBE3  8E 00 FF      lds #TOS      ;load stack at top of ram
45 1917

```

EP 0 549 855 A2

```

1918 FBEB 8D 40      bsr INITS      ;initialize sub-systems
1919
1920 FBEB 0E          cli          ;clear int mask. Int's enabled globally
1921             ;@@@ and other stuff yet
5
1922
1923             ;* signon
1924             ;  idx #SIGNON    ;say hello
1925             ;  jsr PMSG
1926             ;SIGNON text is only version number since we've run out of space
1927             ;  ldaa #VER      ;current version
1928             ;  jsr HOUTS      ;output
10
1929
1930 FBEB CE F8 00      lda #DATE      ;Rule date in EE
1931 FBEC C8 03          lda #3        ;this many
1932 FBEE BD FE 88      jsr PHMSG      ;saved in hex
1933
1934             ;ifdef ILLBUG
1935 FBF1 DE 44          ldaa Msbb      ;crashed?
1936 FBF3 27 0D          beq ?ncrsh  ;no
1937
1938 FBF5 96 20          ldaa Scnrl      ;last scanned rule
1939 FBF7 BD FE AF      jsr HOUTS      ;inform
1940
20
1941 FBFA C8 10          lda #16      ;this many out
1942 FBFC CE 00 AA      lda #IN_BUFS TA ;from here!
1943 FBFF BD FE 88      jsr PHMSG      ;inform user
1944 FCO2 ?ncrsh:        ;endif
1945
1946
25
1947             ;* power on start command
1948             ;  ldaa lstcmd      ;last entered command saved in eeprom
1949             ;  ldaa #'7'      ;starting cmd
1950             ;  jsr OUT_OC      ;echo
1951
1952             ;  bra TSTCMD      ;fake it
30
1953
1954             ;*****
1955             ;MAIN program command loop
1956             ;*****
1957 FCO2 BD FE 60      lpin: jsr Iprompt      ;new line & rule # on screen
1958 FCO5 12 41 08 F9      lpin: bset Flag2,CFLG,lpin      ;Rule changed? print it
35
1959
1960             ;  sei          ;Wait for any interrupt (SCI desired)
1961 FCO9 BD FF 21      jsr IN_DQ      ;Get char?
1962
1963 FCOC 25 F7          bcs lpin      ;Wait until we have one
1964
40
1965 FCOE 13 40 40 09      brcr Flag1,EFLG,necho      ;Echo? none if cr
1966
1967 FC12 81 00          cmpa #CR
1968 FC14 26 02          bne echo
1969
1970 FC16 20 EA          bra lpin      ;crlf & prompt
1971
45
1972 FC18 BD FE 3E      echo: jsr OUT_OC      ;Echo it
1973
1974 FC1B               necho:

```

50

55

			TSTCMD:	
	1975	FC18	?cmds:	Idx #CMDTBL ;Use this table
	1976	FC1B CE F9 00		jsr CMDDISP ;CoMmand DISPatch
	1977	FC1E BD F9 32		bcs ?err3 ;Command not found
	1978	FC21 25 02		
	1979			
5	1980	FC23 20 00		bra lpin ;Get next command
	1981			
	1982	FC25 88 21	?err3:	ldaa #'!' ;Bad command response
	1983	FC27 BD FE 3E		jsr OUT_QC ;Send it
	1984	FC2A BD FE 3E		jsr OUT_QC ;Twice
	1985			
10	1986	FC2D D6 A4	?ewt:	ldab Out_bffil ;Has everything been printed?
	1987	FC2F 26 FC		bne ?ewt ;Hang around until it is
	1988			
	1989	FC31 20 CF		bra lpin ;Start over
	1990			
	1991	FC33 0C	NULL:	clc ;stop errors
15	1992	FC34 39		rts ;Null cmd input characters jsr to here
	1993			
	1994			.page

20

25

30

35

40

45

50

55

EP 0 549 855 A2

```

1995      ;**** subsystem initialize ****
1996 FC35      INIT$:
1997      ;@@ could do this with a LDD, SDD if space gets tight
5  1998      ;  idea #EFLG      ;start with...
1999      ;no XOFFR, ECHO, . . . , no HEX INPUT
2000      ;  staa Flag1      ;stuff it
2001
2002      ;  idea #0      ;nothing here for now
2003      ;  staa Flag2
2004 FC35 CC 40 00      ldd #((EFLG)*256+0)      ;combined. Flag1*256 + Flag2
10  2005 FC38 DD 40      std Flag1      ;stuffed here
2006 FC3A
2007 FC3A DD 42      std Flag3      ;rule Flag3 starts out CLEAR (Flag4 too)
2008
2009 FC3C CC 00 40      ldd #Tmot      ;dummy address initially
2010 FC3F DD 48      std Adjadr      ;stuffed here for tune
15  2011
2012 FC41 F6 F8 05      ldd MSAFE      ;an initial pulse width from EE
2013 FC44 D7 40      std Tmot      ;to protect motor.
2014
2015 FC46 8D 41      bsr SCIINI      ;startup sci, X will point to REG base on exit
2016
2017 (01)      ifndef CHIPA2
2018      jsr ERUDATE      ;update EE if required
2019 (00)      endif
2020
2021      ;  ldx #Filblk      ;block to fill
2022      ;  ldd #$0010      ;Fill block, length
2023      ;?flip:      sta 0,x      ;stuff fill value
25  2024      ;  inx
2025      ;  decb      ;count down
2026      ;  bne ?flip      ;until done (256 max)
2027      ;12 bytes
2028
2029      ;  ldd #0000      ;zero
30  2030 FC48 4F      clra      ;Zeros
2031 FC49 5F      clrb      ;the small way to get D=0
2032 FC4A DD 10      std Srcadr      ;no errors on startup
2033 FC4C DD 70      std Mtime      ;no pulses at start
2034 FC4E DD 16      std Stime      ;no stop time
2035      ;#  std Btime      ;no false beeper
35  2036
2037      ;@ maybe Curru = $FF better?
2038 FC50 5A      decb      ;turn into rule into $FF
2039 FC51 DD 18      std Curmod      ;start with MODE00 (A) & this rule (B) on power up
2040
2041      ;this is where the startup rule is fired
40  2042 FC53 CC 01 01      ldd #$0101      ;force rule 1 on 1st timint (#$tt)
2043 FC56 DD 1C      std Frcurl      ;by stuffing here
2044
2045      ;  ldd #$0001      ;# no repeat, .2 beep on reset
2046 FC58 DD 3D      std Btime      ;# beep .2 sec on startup
2047
45  2048 FC5A CE 10 00      ldx #REG      ;get base again (EEUDATE may corrupt)
2049      ;  idea #300      ;set all bits as input
2050      ;  idea #FF      ;set all bits as output for now (no pullups)
2051      ;  staa DDRC      ;Data DiR C

```

50

55

## EP 0 549 855 A2

```

2052 FC50 6F 07      clr _ODRC,X      ;all inputs on C
2053
2054 FC5F 86 83      ldd #$83       ;PA1 is out, RTInt @ 32.77mS if enabled
2055 FC61 A7 26      staa _PACTL,X  ;config A ports bit & Real Time Int rate
5
2056
2057      ;spini:
2058 FC63 C6 FF      ldd #$FF       ;nearly all bits output (*SS is gen purpose out)
2059 FC65 E7 09      stab _DORD,X  ;set port D direction
2060
2061 FC67 C6 51      ldd #$601010001 ;SPIE,SPE,DPSH/PUL,MSTR CPOLH,CPHA0,SPR1&0
2062 FC69 E7 28      staa _SPCR,X  ;Enable SPI as Master @ E/4 baud
10
2063
2064      ; ldd _SPSR,X  ;Fake a read to clear SPIF flag
2065      ; ldd _SPDR,X  ;so writes possible
2066
2067      ;*****
2068      ;startup timer OC1
2069      ;we won't do this here since we don't want a restart to be visible, eh?
2070      ;1st OC1 int 32msec after power up???
2071      ; ldd TCNT      ;currently
2072      ; addd #RTCRAT  ;some time from now
2073      ; std TOC1      ;setup
2074
2075 FC6B 86 80      ldd #$80       ;OC1 bit
2076      ; staa _OC1M,X  ;let OC1 affect OC1 bit
2077      ; staa _OC1D,X  ;set OC1 high on compares, others low
2078      ;done earlier  staa _PACTL,X  ;enable output driver OC1
2079 FC6D A7 22      staa _TMSK1,X ;enable OC1 timer ints ($80)
2080
2081      ;@@@ this should be under interrupt (and maybe power down if slow enough)
2082      ; Start A/D scanning/converting
2083 FC8F 86 30      ldd #$30       ;CCF/x/scan/mult ch0-4 :scan 0-4 repeatedly
2084      ; ldd #$10       ;scan 0-4 once
2085 FC71 A7 30      staa _ADCTL,X ;start A/D
2086
2087      ;*** COPYR copies MODE00 to ram
2088 FC73 CE FB 16      COPYR: ldy #MODE00  ;src here
30
2089 FC76 18 CE 00 12  copybf: ldy #Curmbf ;dest here
2090 FC7A F8 F8 0B      ldd BFCNT    ;bit field size
2091      ;fall into
2092      ;*****
2093      ;COPYM. X points to Src, Y to Dest, B is size. Called everywhere
2094      ;*****
35
2095 FC7D A6 00      COPYM: ldd O,x ;get data
2096 FC7F 18 A7 00      staa O,y ;stuff here
2097 FC82 08      inx          ;bump up pointers
2098 FC83 18 08      iny
2099 FC85 5A      decb         ;count down
2100 FC88 2E F5      bgt COPYM   ;until done
2101
40
2102 FC88 39      rts
2103      ;*** end of inits
2104
2105      ;*****
2106      ;SCIINI. Initialize HDPTR, TL PTR, BFSIZ , BAUD, SCCR's
2107      ;*****
2108      ;could use copym but only one byte saved 18L vs 17M
45

```

50

55

EP 0 549 855 A2

```

2109 FC89 CC 00 AA      SCIINI: ldd #IN_BUFGSTA ;init hdptrs & dptrs to start of buffers
2110 FC8C DD A6          std In_hdptr
2111 FC8E DD A8          std In_dptr
2112
5   2113 FC90 CC 00 80      ldd #OUT_BUFGSTA
2114 FC93 DD A0          std Out_hdptr
2115 FC95 DD A2          std Out_dptr
2116
2117 : ldd #0000          ;buffers start out empty
2118 FC97 4F              clra          ;buffers start out empty
10   2119 FC98 5F          clrb          ;the smart way to get D=0
2120 FC99 DD A4          std Out_bffil ; which clears In_bffil in following byte too
2121
2122 : Initialize SCI registers
2123 FC9B CE 10 00          ldx #REG          ;register base
2124 : ldd #300             ;1/8/1 bits. Idle line wakes rcr
15   2125 : staa SCCR1
2126 FC9E 6F 2C          clr _SCCR1,X ;setup
2127
2128 : EE byte for baud rate
2129 : ifndef CHIPA2
2130 : ldd #F803            ;from here
2131 : else
20   2132 FCA0 B6 F8 03      ldd #BDRATE      ;430 = E/13/1 > 9600 $34 = E/13/16 > 600 baud
2133 : endif
2134
2135 : staa BAUD
2136 FCA3 A7 2B          staa _BAUD,X
2137
25   2138 : SCIIEN:          ;EEUDATE/SAVE returns here to restart Rxint's
2139 FCA5 CE 10 00          ldx #REG          ;base again because of EE (6 vs 8)
2140 FCA8 1C 2D 2C          beq _SCCR2,X,$2C ;RcvIntE,TxEn,RxEn
2141 FCAB 39              rts
2142 : ***** end of scii
2143 : .page
30

```

35

40

45

50

55

```

2144      ;*****
2145      ; EE UpDATES only changed bytes since faster than whole rewrite
2146      ; Copies a piece of code to RAM at IN_BUFSTA since EE disappears on EELAT
2147      ;*****
5       2148      ;@@@ we now only update a small block at a time
2149      ;svmsg: db "avin","g"+$80
2150
2151      ;***** SAVE; updates EE
2152 FCAC      SAVE:
2153      ; bsr EEUDATE      ;called at command level now
2154      ; bra SCIIEN      ;restart Rxints
10      2155 FCAC      EEUDATE:
2156      ; ldx #svmsg      ;inform
2157      ; jsr PMSG      ;user
2158
2159      ;@@ we should be so fast that no one notices 10msc hick-up
15      2160      ; sei      ;turn off ints & pray for no COP, XIRQ
2161
2162      ;@ strictly speaking RXint ONLY needs to be off at this point. Later ALL off.
2163 FCAC CE 10 2D      ldx #SCCR2      ;register to mess with
2164 FCAF 1D 00 20      bclr 0,x,$20      ;kill rxint enable (6 bytes)
2165
20       2166 FCB2 CE FD 00      ldx #pbcc      ;src
2167 FCB5 18 CE 00 AA      ldy #IN_BUFSTA      ;dst. since usually empty
2168 FC89 C8 10      ldeq #pbcc-pbcc      ;length less than 256! (16 actually)
2169
2170 FCBB BD FC 7D      jsr COPYM      ;copy EE subroutine to ram
2171
25       2172 FCBE DE 10      ldx Srcadr      ;current source in ram (EE write adr)
2173      ; beq pbret      ;nothing to save, AN ERROR
2174 FCC0 8C F8 00      cpx #EE      ;where EE sits
2175 FCC3 25 31      blo pbret      ;not EE, skip it!
2176
2177 FCC5 18 CE 00 00      ldy #Wrkrul      ;ram table
30       2178 FCC9 C8 10      ldeq #RSIZ      ;is this big
2179 FCCB 18 A6 00      pcomp: ldeq 0,y      ;get current ram data
2180 FCCE A1 00      cmpe 0,x      ;compare with previous saved EE
2181 FCDD 27 1E      beq bdec      ;same so skip pgmng this byte
2182      ;
2183 FCD2 37      pshb      ;save counter
35       2184 FCD3 18 3C      pshy      ;save current pointer
2185      ;dbg...
2186 FCD5 36      psha      ;save data
2187 FCD6 BD FE A8      jsr HOUTC2      ;dbg
2188
2189 FCD9 32      puls      ;get back
40       2190 FCDA 36      psha      ;save again
2191 FCDB BD FE AF      jsr HOUTS      ;dbg
2192
2193 FCDE 32      puls      ;get data back
2194      ;...dbg
2195 FCDF C8 16      ldeq #$16      ;byte erase first
45       2196 FCE1 8D 15      bsr pbe      ;do it
2197
2198 FCE3 0E      cli      ;allow an int (but no rxint) since rules still in Ram
2199 FCE4 81 FF      cmpe #$FF      ;did we only need an erase?
2200 FCE6 27 04      beq ffdone      ;yes

```

EP 0 549 855 A2

```

2201
2202 FCE8 C6 02           ldat #$2 :program
2203 FCEA 80 0C           bsr pbe      ;now
2204
2205 FCEC 0E           ttdone: cli      ;allow a few more ints
5 2206 FCF0 18 38           puly      ;recover em
2207 FCFE 33           pulb
2208
2209 FCF0 08           bdec: inx      ;bump pointers ahead
2210 FCF1 18 08           inv
2211 FCF3 5A           decb      ;and count down
2212 FCF4 26 D5           bne pcomp ;until zero
10 2213
2214 FCF6           pbrst:
2215 FCF6 20 AD           bra SCIIEN ;restart Rxints
2216           rts      ;done update
2217
2218 (01)           rdef COPON ;copy enabled
15 2219           :copset resets the COP timer
2220           copset: ldat #$55 ;COP needs attention
2221           stab COPRST
2222           comb      ;flip it to $AA, one byte shorter
2223           stab COPRST
2224           rts
2225
20 2226           ;**** code compression subr
2227           pbe: bsr copset ;reset cop
2228
2229 (01)           pbe: else
2230 FCF8           pbe: endif
2231 (00)
2232
25 2233 FCF8 18 CE 08 98           ldy #2200 ;2500 = 10 msec@ 4cy/day 3333 = 10msec@3/dex + 3/bne
2234 FCF0 0F           sei      ;stop ALL ints & pray no XIRQ
2235           jmp pbe: ;program byte/erase & ret
2236 FCFD 7E 00 AA           jmp IN_BUFSTA ;its really here
2237
30 2238           ;*** the following 16 bytes get copied to ram so EEUDATE can call with
2239           ;B set to erase or pgm cmd, Y with delay value, and X pointing to address.
2240           ;A with value to pgm
2241 FDO0           pbe:
2242 FDO0 F7 10 3B           stab PPROG ;set EELAT
2243 FDO3 A7 00           staa 0,x ;write or erase @ x
2244 FDO5 7C 10 3B           inc PPROG ;EEPROM up
35 2245 FDO8 18 09           wt10: dev      ;count down
2246 FDOA 28 FC           bne wt10 ;until done
2247 FD0C 7F 10 3B           clr PPROG ;finished
2248 FD0F 39           rts      ;return to EE code
2249 FD10           pbe: equ $ ;end of part copied to ram. 16 bytes
2250           ;*** end of EEUDATE
2251           .page
40

```

45

50

55

```

2252      ****
2253      ;ENTER; Fiddles memory locations. Operates directly if ADDR < $8000
2254      ;ADJUST; Adjust values of rule table. Asks for rule #, copies to ram if not
2255      ;already there, and then asks for parameter to change.
5       2256      ;@ have fun...
2257      ****
2258      :evmsg: db "Enter ", "@", $80
2259      :rnmsg: db "dj ", "#", $80
2260      FD10 0D 0A C5      :lmsg: db CR,LF, "E", $80      ;"Elem", "#", $80
2261      :nvmsg: db "Ne", "w", $80
10      2262      ****
2263      FD13      update:      cpx Srcadr      ;same as current request in X?
2264      FD13 9C 10      beq ?endud      ;yes, no RAM update required
2265      FD15 27 0D
2266
2267      ;@ do we want to prompt for save? Do we want a reread?
2268      ;      jsr EEUPDATE      ;different, need to update EE before overwrite
15      2269      FD17
2270      FD17 DE 44      idx Msbb      ;recover requested source address
2271      FD19 DF 10      stx Srcadr      ;new set will be from here
2272      FD1B 18 CE 00 00      idx #Wrkrul      ;destination is...
2273      FD1F C6 10      ldb #RSIZ      ;this big
2274      FD21 8D FC 7D      jsr COPYM      ;copy new set to ram
20      2275
2276      FD24 DE 10      ?endud:      idx Srcadr      ;get it again (unfortunately)
2277      FD26 DF 25      stx Offadr      ;fix offset for print outs
2278      FD28 CE 00 00      idx #Wrkrul      ;this is where data now (or still) sits
2279      FD2B DF 4B      stx Adjadr      ;setup work pointer
2280      FD2D 39      rts
25      2281      **** ENTER |      ;uses absolute address, no rule fiddling
2282      ;ENTER:
2283      FD2E      ;      idx #evmsg      ;enter value message
2284      ;      bsr GETNUM      ;input an address. Returns to main if nothing entered
2285      ;      bsr GETVAL      ;get a value. Return to main if none.
2286      FD2E 8D 67
287
30      2288      FD30 CE 01 10      idx #S0110      ;header 1 line, 16 elements
2289      FD33 BD FE D6      jsr ENTDMP      ;do header & data @ Msbb
2290
2291      FD36 4F      clra      ;zero
2292      FD37 5F      clrb      ;the small way to get D=0
2293      FD38 DD 25      std Offadr      ;assume direct addr. Clear offset
35      2294
2295      FD3A DE 44      idx Msbb      ;get the address to fiddle
2296      FD3C 8C B8 00      cpx #S8600      ;lower limit of EE
2297      FD3F 25 2E      blo fuladr      ;no EE update, use it directly.
2298
2299      FD41 8D D0      bsr update      ;update EE if different from current
40      2300
2301      FD43 20 2A      bra fuladr      ;to common code
2302
2303      **** ADJUST; twiddle rules, friendlier user interface
2304      FD45      ADJUST:
2305      ;      idx #rnmsg      ;rule # msg
2306      ;      bsr GETNUM      ;input rule #. Returns to main if nothing entered
45      2307      FD45 8D 50      bsr GETVAL      ;get a value. Return to main if none.
2308

```

## EP 0 549 855 A2

```

2309 FD47 D6 45      lddb Lsbb      ;get rule # into B
2310 FD49 37          pshb          ;save for 2nd rule difference
2311 FD4A BD FA 3A      jsr GETEDR    ;get rule's address from indexed table (we think)
2312                   bcs ?nrule  ;already in RAM
5
2313                   stx Adraddr  ;save it here for later use
2314                   stx Msbb      ;& stuff for memdump to use
2315 FD40 DF 44          bsr update   ;update EE if needed
2316 FD4F 8D C2
2317
2318 FD51 33      ?nrule:  publ      ;recover rule #
2319 FD52 5C          incb          ;next rule #
10
2320 FD53 BD FA 3A      jsr GETEDR    ;get next rule's address
2321                   ;bcs not needed here since if rule was in ram, this one wouldn't be
2322                   ;and if it wasn't, rule would have been put there by update
2323
2324 FD56 8F          xgdx          ;into D
2325 FD57 93 10          subd Srcaddr  ;find address difference
15
2326                   bne ?ednz    ;address difference NOT zero, may be ok
2327
2328                   ;if difference =0 then same rule at both locations. Do we get the NEXT one up &
2329                   ;try again?? For now, just limit to 16 elements.
2330 FD59 C4 0F          andb #$0F      ;and B has length (16 max)
2331 FD5B 28 02          bne ?bnz     ;reasonable length?
20
2332
2333 FD5D C6 10      ?bnz:  lddb #$10      ;limit to 16
2334 FD5F 86 01          lddb #1       ;one line of info, b has length
2335 FD61 8F          xgdx          ;swap to X
2336 FD62 BD FE CB      jsr ADJDMP    ;let memdump do the work
2337
25
2338 FD65 CE FD 10      ldx #elmsg   ;element #
2339 FD68 BD 2A          bsr GETNUM   ;get info, returns to main if no entry
2340
2341                   ldd Adraddr  ;recover basic address
2342                   addd Msbb     ;include element offset (16bit)
2343                   xgdx          ;into X
30
2344                   ;since we can't be more than 16 elements offset
2345 FD6A DE 48          ldx Adraddr  ;get basic address in RAM!!
2346 FD6C D6 45          lddb Lsbb    ;include element offset
2347 FD6E 3A          ebx          ;include in X
2348 FD6F BD 12          fuladdr:  bsr prnaddr ;tell them
2349
2350 FD71 20 3C          bra tadj    ;tune it
35
2351
2352                   ;RTDSPH uses these. X preset to Adraddr by CMDSPH
2353 FD73 SETADR:        ;SETADR:
2354 FD73 BD FF 3D      jsr HEX_BIQ   ;get an address
2355 FD76 25 26          bcs getret'  ;no input, use old value (RTS)
2356
40
2357 FD78 4F          cirb          ;the small way to get D=0
2358 FD79 5F          cirb
2359 FD7A DD 25          std Offaddr ;clear offsets
2360
2361 FD7C DE 44          ldx Msbb     ;get input value
2362 FD7E 20 03          bra prnaddr ;stuff & print
45
2363
2364 FD80 INCADR:        ;INCADR:

```

50

55

EP 0 549 855 A2

```

2365 FD80 08      inx      ;modify pointer
2366 FD81 08      inx      ;compensate for falling into dex
2367 FD82
2368 FD82
2369 FD82 09      DECADR:    dex      ;modify pointer
5      2370
2371      ;limit the address to modulo 16
2372      ;    xgdx      ;swap to D for limits
2373      ;    clra      ;high order of address < 256 (RAM @ 0)
2374      ;    andb #3F    ;modulo 16. A will be zero
2375      ;    xgdx      ;back to X
10      2376      ;fall into ...
2377
2378 FD83 0F 4B      prnadr:  stx Adjsdr  ;save for RTDSPH functions
2379 FD85 3C      pshx      ;save Ram address
2380 FD86 8F      xgdx      ;swap into D
2381 FD87 D3 25      addd Offadr  ;equivalent address offset (since data's at zero now)
2382 FD89 8F      xgdx      ;back to X
15      2383
2384 FD8A BD FE A8      jsr HOUTC2  ;print it
2385
2386 FD8D 38      pulx      ;recover ram address
2387 FD8E BD FE A2      jsr CRLF  ;new line &
2388
20      2389 FD91 7E FF 0C      jmp svp      ;prompt for adjustments
2390
2391      ;*****
2392      ;GETNUM. prompts with message @ X & gets a number with echo. Returns to main
2393      ;no entry.
2394      ;GETVAL As above but no message.
2395      ;Placed here so BSR's in possible
25      2396 FD94 BD FE 71      GETNUM:   jsr PMSG      ;prompt user
2397
2398 FD97 BD FF 3D      GETVAL:   jsr HEX_BIQ  ;get value
2399
30      2400      ;    bcs NULL      ;no entry, return directly
2401 FD9A 24 01      ;    bcc ?getok      ;no error, normal return
2402
2403 FD9C 38      pulx      ;pop callers address. Returning to MAIN      now
2404 FD9D      ?getok:
2405      ;    idx Msbb      ;get value
2406      ;    jmp HOUT2S  ;echo it
35      2407
2408 FD9D 0C      clc      ;make certain no error to MAIN
2409 FD9E 39      getret:  rts      ;also used by SETADR
2410
2411      ;*****
2412      ;TUNE; write values to the motor
2413      ;@ Could we use adjust functions??
2414      ;*****
40      2415      ;tnmsg: db "une", " + $80
2416 FD9F      TUNE:
2417      ;    idx #tnmsg
2418      ;    jsr PMSG      ;tune msg
2419
45      2420 FD9F 14 41 01      bset Flag2,TFLG  ;stop state diagram, set tuning flag

```

EP 0 549 855 A2

```

2421
2422 FDA2 CE FD C7      :Idx #texit      ;tune exit
2423 FDAS 3C             pshx      ;setup as return for RTDSPH if ESC entered
2424
2425 ::      :Idx #Adjadr      ;last address adjusted
5  2426 ::      :ldaa 0.x      ;get value
2427 ::      :staa Tmot      ;save here
2428
2429 FDA6 CE 00 40      :Idx #Tmot      ;address to fiddle
2430 FDA9 DF 48          stx Adjadr ;setup for keyboard
2431
10 2432 FDAB 4F          clra
2433 FDAC 5F          clrb      ;the small way to get D=0
2434 FDAD DD 25          std Offadr ;Zero offset for correct print out
2435
2436      ;entry here from enter and adjust
2437 FDAF BD FE A2      tadj: jsr CRLF ;new line for echos
15  2438 FDB2 BD FF 21      ?sip: jsr IN_DQ      ;get a char
2439 FDB5 25 FB          bcs ?sip ;none avail, loop here
2440
2441 FD87 BD F9 2F      jsr RTDSPH      ;adjust a variable
2442 FDBA 25 F6          bcs ?sip ;not one of ours so loop again
2443
20  2444 FDBC 13 41 01 F2      brcr Flag2,TFLG,?sip      ;not tuning motor, skip motor update code
2445
2446 FDC0 96 4D          ldaa Tmot      ;from new temporary value
2447 FDC2 BD FB 76          jsr LD_MOTOR ;scale to Mtime
2448 FDC5
2449 FDC5 20 EB          bra ?sip ;and hang around until ESC causes exit
2450
25  2451      ; entered by escexit's attempt to return to main
2452 FDC7      texit:      brcr Flag2,TFLG ;reenable state machine
2453 FDC7 15 41 01      ; adjust's abort exit too
2454
2455 FDCA      ended:      rts      ;return to main
30  2456 FDCA 39
2457      .page

```

35

40

45

50

55

EP 0 549 855 A2

```

2458      ;*****
2459      ;Serial communication interrupt routine
2460      ;*****
2461      ; Need to enable ints earlier. Clear rxint & txint for each routines section
2462      ; also need a get char function that doesn't move tpntr so editing possible!
5
2463      ;@ Needs to be sped up. @9600B less than 1msec/char available, + other ints!
2464  FDCB      SCIIINT:
2465  FDCB  CE 10 00      ldx #REG      ;base
2466      lddaa SCSR      ;who interrupted?
2467  FDCE  A6 2E      lddaa _SCSR,x      ;who interrupted?
2468  FDD0  85 20      bits #320      ;Tx/Tc/Rdrf/lcl Or/Nf/Fc/X
2469  FDD2  27 36      beq trantst      ;is NOT the receiver, maybe tx
2470
2471      lddaa SCDR      ;IS receiver, so get character
2472  FDD4  A6 2F      lddaa _SCDR,x      ;IS receiver, so get character
2473  FDD6  84 7F      anda #7f      ;clear parity
2474
15
2475  FDD8  81 13      cmpe #XOFF      ;is it XOFF?
2476  FDDA  26 05      bne ?xont      ;nope
2477
2478  FDDC  14 40 80      bset Flag1,XFRFLG      ;stop further tx. Xoff received
2479  FDDF  20 52      bre notran      ;stop immediately
2480
20
2481  FDE1  81 11      ?xont: cmpe #XON      ;:XON?
2482  FDE3  26 08      bne wt_in      ;:nope, must EN_Q it
2483
2484  FDE5  15 40 80      bcir Flag1,XFRFLG      ;XON, let tx go
2485  FDE8
2486  FDEB  96 A4      lddaa Out_bffil      ;stuff avail? (lda shorter than tst)
2487  FDEA  27 3F      beq endint      ;no, so fini, nothing to TX now that we're allowed
2488
2489  FDEC  8D 41      bar txon      ;startup tx ints
2490
2491  FDEE  20 1A      bra trantst      ;and process pending int
2492
2493
30
2494      ; Queue the character in acca --> input buffer
2495      ; change to 8 bit offsets for speed/size? ABX with b as offset
2496      ;*****
2497      ;@ this is a TRAP. Can't get characters out since we're hung waiting
2498      ;@ in an ISR!!!
2499  FDF0  D6 A5      wt_in: lddab In_bffil      ;get current size
35
2500  FDF2  C1 11      cmpb #IN_BFMAX      ;check for room
2501
2502      ;* send XOFF if near full
2503
2504      ; bhi wt_in      ;we'll have to wait for room
2505  FDF4  22 14      bhi trantst      ;we'll just have to loose char while we wait for room
40
2506
2507      ; could load bfmax into b and loop on cmp bffil for tighter loop
2508      ;@could also allow 2 chars more into buffer until xoff takes effect
2509  FDF6  3C
2510  FDF7  DE A6      pshx      ;save reg pointer
2511  FDF9  A7 00      ldx In_hdptr      ;get head pointer
2512
2513  FDFB  7C 00 A5      staa O,X      ;and store char in que
45
      inc In_bffil      ;one more in queue, safe from int's

```

EP 0 549 855 A2

```

2514
2515 FDFE 08           inx      :step pointer ahead
2516 FDFF 8C 00 BF      cpx #IN_BFEND :past end?
2517 FE02 23 03          bis in_norapl :no, no wrap needed
5  2518 FEO4 CE 00 AA      ldx #IN_BUFSTA :yes, wrap back to the start of buffer
2519
2520 FE07               in_norapl:
2521 FE07 DF A8          stx in_hdptr :update hdptr
2522 FEO9 38              pulx    :recover reg pointer
2523
10   2524               ;fall through to..
2525
2526   ;***+
2527   ;txint serviced here after rx does work (also rx bra's if XON/XOFF)
2528   ;***+
2529 FEOA               transt:
15   2530 FEOA A6 2E      ldeaa _SCSR,X :is it the transmitter?
2531 FEOC 2A 1D          bpl endint :no, so we're done poll (TXBE = MSB, high is neg)
2532
2533 FEOE 12 40 80 21      breset Flag1,XFRFLG,notran :Xoff? Yes, turn off txint for now
2534
2535 FE12 96 A4          ldeaa Out_bffil :is there anything left to transmit?
2536 FE14 27 1D          beq notran :empty, so skip to end & kill future ints
20
2537
2538 FE16 DE A2          ldx Out_tptr :get pointer
2539 FE18 A6 00          ldeaa O,x :get char
2540 FE1A B7 10 2F          staa SCDR :send it out
2541 FE1D
2542 FE1D 08           inx      :skip to next spot
2543 FE1E 8C 00 9F          cpx #OUT_BFEND :did we fall off end of buffer?
2544 FE21 23 03          bis out_nowrp2 :no, skip wrap
2545
2546 FE23 CE 00 80      ldx #OUT_BFSTA :yes, point back to the start
2547
2548 FE26               out_nowrp2:
30   2549 FE28 DF A2      stx Out_tptr :update tail pointer
2550 FE28 7A 00 A4      dec Out_bffil :one less occupied space
2551
2552   :: cmpa #OUT_BFMIN :should we send XON character?
2553   :: bhi out_xon :no
2554
35   2555               ;Send XON, space available
2556   ;:out_xon:
2557
2558 FE2B 3B           endint: rti      ;return later
2559
2560   ;**** called by rxint for Xon, and OUT_QC :> A must be preserved
40   2561 FE2C CE 10 00      txonx: ldx #REG :for out_qc
2562 FE2F 1C 2D 80      txon: bset_SCCR2,x,$80 :set Tx Int En if not already up
2563
2564 FE32 39           rts
2565
2566
45   2567 notran: ;@ bclr?? :entered by rxint XOFF
   ; ldeaa SCCR2 :get current flags
   ; anda #37f :Disable transmit interrupts
   ; staa SCCR2 :by lowering enable
2568
2569
2570

```

## EP 0 549 855 A2

```

2571          :Idx #REG      :pointer
2572 FE33 1D 2D 80      :bclr _SCCR2.x,$80      :disable tx int's
2573
2574          XRET:  ;xirq rti
5  2575 FE36 3B          rti      ;all done here
2576
2577          ;***** here for bsr limits
2578 2578 FE37 0D 0A 20 20 20      spcmmsg: db CR,LF,"    "    > $80      ;blanks for memdmp
      FE3C 20 A0.
2579          ;*****
10  2580          ;Queue the character in acca --> output buffer
2581          ;Returns A unchanged
2582          ;:: change to 8 bit offsets from x to improve speed/size since only 256 RAM
2583          ;:: use B more !
2584          ;Preserves B & X & A
2585          ;*****
2586 2586 FE3E
15  2587 FE3E 37          pshb      ;Save bytes elsewhere
2588 FE3F 3C          pshx      ;since we're doing this a lot
2589
2590 FE40 D6 A4          wt_out: ldab Out_bffil      ;get current size
2591
2592          ;@ @ @ should probably be a ram location
2593 2593 FE42 C1 1F          cmpb #OUT_BFMAX      ;check for room
2594 FE44 22 FA          bhi wt_out      ;we'll have to wait for room
2595          ;:: or load ac with max and loop on compare bffil for tighter loop
2596          ;::perhaps set carry and return instead of looping
2597
2598 2598 FE46 DE A0          idx Out_hdptr      ;get head pointer
2599 FE48 A7 00          staa 0,X      ;and store char in que
25
2600 2601 FE4A 7C 00 A4          inc Out_bffil      ;one more in queue ;int's can't affect inca
2602
2603 2603 FE4D 08          inx      ;step pointer ahead
2604
2605 2605 FE4E 8C 00 9F          cpx #OUT_BFEND      ;past end?
2606 2606 FE51 23 03          bts out_norapl      ;no, no wrap needed
30  2607
2608 2608 FE53 CE 00 80          idx #OUT_BFSTA      ;yes, wrap back to the start of buffer
2609
2610 2610 FE56 DF A0          out_norapl:
2611 2611 FE56 0F A0          stx Out_hdptr      ;update hdptr
35  2612
2613 2613 FE58 8D D2          bsr txonx      ;start up ints now that X can be mangled
2614
2615 2615 FE5A 38          pulx      ;get it all back
2616 2616 FE5B 33          clc      ;prevent errors
2617 2617 FESC 0C
2818
2619 2619 FE5D 39          rts      ;of OUT_QC
2620          .page

```

45

50

```

2621      ;*****
2622      ;FREMOD:    bclr Flag2,MFLG  ;clear mode locked flag
2623      ;      idx #?frmmsg      ;and inform
2624      ;      brc mend      ;common end
5       2625
2626      ;?frmmsg: db "re","e"+$80
2627      ;!frmmsg: db "ocke","d"+$80
2628
2629      ;*****
2630      ;LOCMOD:    bset Flag2,MFLG  ;set mode locked flag
10      2631      ;      idx #!frmmsg
2632      ;!mend:  ;fall into PMSG
2633      ;      bra PMSG
2634      ;**** some of the commands called from RTTBL
2635      ;Esc exit. CMDDISP jumps to here if ESC entered
2636  FE5E  ESCEXIT:
15      2637      ;      pulx      ;pop RTDSPH return (2 not used if BRA cmdsp)
2638  FE5E  38      pulx      ;pop caller return (Tune, Dump, etc)
2639
2640  FE5F  39      rts      ;should land us in main
2641
2642      ;*****
2643      ;puts up new line prompt
20      2644  FE60  lprompt:
2645  FE60  8D 40      bsr CRLF      ;New line on screen
2646
2647  FE62  96 18      ldax Curmod      ;get current mode
2648  FE64  8D 59      bsr OUTRH      ;right hand nibble out
2649
25      2650  FE66  96 19      ldax Currl      ;current new rule
2651  FE68  8D 45      bsr HOUTS      ;print rule
2652
2653  FE6A  15 41 08      bclr Flag2,CFLG  ;clear current rule needs print flag
2654
30      2655  FE6D  86 2A      LDAA #PROMPT ;Prompt character
2656  FE6F  20 CD      bra OUT_QC      ;Stuff in que & return
2657
2658      ;***** end lprompt
2659
2660      ;*****
35      2661      ;PMSG. print string @ X, null or hi-bit terminated
2662      ;BSRs OUT_QC
2663
2664  FE71  PMSG:
2665      ::      tpe      ;get CC
2666      ::      pshe      ;save. others may need C bit info
2667
40      2668  FE71  A6 00      ldax O,x      ;get char
2669  FE73  27 09      beq ?prnd      ;zero is end
2670
2671  FE75  84 7F      andx #37F      ;clear high bit to get rid of funny IBM chars
2672  FE77  8D C5      bsr OUT_QC      ;print it (PMSG)
2673
45      2674  FE79  8D 00      tst O,x      ;check for end, MSB high
2675  FE7B  08      inx      ;point to next byte
2676  FE7C  2A F3      bpl PMSG      ;more if MSB =0
2677  FE7E  ?prnd:

```

50

55

EP 0 549 855 A2

2678		clc	;stop possible errors (tst cleared C)	
2679				
2680	::	pula		
2681	::	tap	;restore CC's	
2682		@CC save/restore currently bombs. WHY?		
2683				
2684	FE7E 39	rts	;all done	
2685				
2686		***** CR then CNTRO		
2687		;CRCNTR: bsr CRLF	;newline	
2688				
2689		***** counts up # of values in B, printing as it goes		
2690		;CNTRO: clra	;start @ zero	
2691	FE7F	CNTR:		
2692	::	pshx	;save	
2693	FE7F 37	pshb	;this too	
2694	FE80	cntrl:		
2695	:	psha	;save	
2696	:	bsr HOUTS	;print it	
2697				
2698	:	decb	;count down	
2699	:	bitb #7	;modulo 7	
2700	:	bne ?nspc2	;no space yet	
2701				
2702	:	bsr OUTS	;cntr spc out	
2703				
2704	:	?nspc2: pula	;recover	
2705	:	inca	;next value	
2706				
2707	:	tstb	;flags	
2708	:	bne cntrl	;not yet	
2709		5 bytes saved. (looked like 10 but bsr/rts/pshx/pulx ate 5)		
2710	FE80 8D 0F		bsr phbc ;subr to print, countb etc, saves a & incs	
2711				
2712	FE82 28 FC		bne cntrl ;B not zero, more to do	
2713				
2714	FE84 33	pulb	;recover	
2715	::	pulx	;too	
2716	FE85 39	rts		
2717				
2718		*****		
2719		;PHMSG. print hex string @ X, space separated, B has length		
2720		;BSRs HOUTS		
2721		;Saves B		
2722		*****		
2723	FE88 8D 1A	PHMSGC:	bsr CRLF	;new line
2724	FE88 37	PHMSG: pshb		;save it
2725	FE89 A8 00		?phmsg: ldax 0,x	;get value
2726	FE88 8D 04			bsr phbc ;call print subr
2727				
2728	FE8D 28 FA		bne ?phmsg	;not done yet
2729				
2730	FE8F 33	pulb	;recover	
2731	FE90 39	rts	;return	
2732				
2733		*** subr for CNTR & PHMSG		
2734	FE91 36	phbc: psha		;save for other caller (cntr)

EP 0 549 855 A2

```

2735 FE92 8D 1B          bsr HOUTS      ;hex out with space (PHM)
2736
2737 FE94 08          inx          ;next up (or wasted cycles if cntr)
2738 FE95 5A          decb          ;count down
5      2739          bitb #$7 ;modulo 7
2740 FE96 C5 03          bitb #33 ;modulo 4
2741 FE98 26 02          bne ?nospc  ;no addition space output
2742
2743 FE9A 8D 15          bsr OUTS      ;add an extra one
2744
10     2745 FE9C 32          ?nospc:  puls          ;recover. the other caller (cntr) needs this
2746 FE9D 4C          inca          ;count up.
2747
2748 FE9E 5D          tstb          ;set flags
2749 FE9F 39          rts          ;return with CCs set
2750
15     2751          ;**** CRLF sends out cr & lf
2752          ;Leaves A as LF
2753          ;**** CRR CR only
2754          ;**** HOUTSC. HOUTS + CRLF
2755 FEA0 8D 00          HOUTSC:  bsr HOUTS      ;print it first
2756          ;fall into
20     2757 FEA2 8D 71          CRLF:   bsr CRR      ;CR first
2758
2759 FEA4 86 0A          ldaa #LF ;LF next
2760 FEA6 20 98          outa:   bra OUT_QC    ;stuff & return (also used by HOUT.. exits)
2761
2762          ;****
2763          ; HOUT2S. Hex OUT 2 Space from X value
2764          ; A returns as SPC. X, B unchanged
25     2765          ; BSRs: HOUT, HOUTS
2766 FEA8 8D F8          HOUTC2:  bsr CRLF      ;newline
2767 FEAA
2768 FEAA 3C          pshx          ;save current value
2769 FEAB 32          puls          ;get high order byte
30     2770 FEAC 8D 07          bsr HOUT      ;print it (H02S)
2771
2772 FEAE 32          puls          ;get low order byte
2773          ; bsr HOUTS      ;print it with space
2774          ;fall into
2775          ;****
35     2776          ; Hex OUT Space. Hex out with space
2777          ; A returned as space, B & X ok
2778          ; JSRs: OUT_QC
2779          ; BSRs: HOUT
2780          ;****
2781 FEAF
40     2782 FEA9 8D 04          HOUTS:  bsr HOUT      ;print it
2783
2784 FEB1 86 20          OUTS:  ldaa #SPC      ;followed by..
2785          ; bra OUT_QC    ;which returns
2786 FEB3 20 F1          bra outa    ;which returns
2787
45     2788          ;****
2789          ; Hex OUT. converts the byte in A to hex and outputs
2790          ; Calls: OUT_QC.
2791          ; Destroys A, B & X ok

```

50

55

EP 0 549 855 A2

2792	*****		
2793 FEB5 36	HOUT:	psha	;save it
2794 FEB6 8D 03		bsr ?outh	;send left half
2795	*****		
5 2796 FEB8 32		pula	;recover
2797 FEB9 20 04		bra OUTRH	;send right half and let IT return
2798	*****		
2799 FEBB 44	?outh:	lsra	;get high nybble first
2800 FECB 44		lsra	
2801 FEBO 44		lsra	
10 2802 FEBE 44		lsra	
2803	*****		
2804 FECF 84 0F	OUTRH:	ands #30F	;mask out high nybble
2805 FEC1 88 30		adds #330	;convert to numeric
2806 FEC3 81 39		cmpa #339	;in 0-9 range?
2807 FEC5 2F DF		ble outa	;yes
2808	*****		
15 2809 FEC7 88 07		adds #07	;bias to A-F
2810 FEC9 20 DB		bra outa	;send & return (hout)
2811	*****		
2812	*****		
2813	*****		
20 2814	; MEMDMP  dumps 256 Memory values at address given		
2815	; Bsr's to HOUT & HOUTS & HOUTC2 &		
2816	; ENTDMP uses Msbb for address, lowX for width, hiX for line count.		
2817	*****		
2818	?lth	equ 16	;line length memdmp
2819 FECB 4F	ADJDMPl	clr a	;start header at zero always
2820 FECC 20 0A		bra memin	
25 2821	*****		
2822 FECE BD FF 3D	MEMDMP:	jsr HEX_BIQ	;get start value ;@@ bsr?
2823 FED1 25 23		bcs mend	;no entry so quit.
2824	*****		
2825 FED3 CE 10 10		Idx #31010	;16 lines, 16 elements
2826	*****		
30 2827 FED6 96 45	ENTDMP:	ldaa Lsbb	;start header count here
2828 FED8 97 48	memin:	staa Acnt	;stuff for later use
2829 FEDA DF 46		stx Lcnt	;set line count & length from X
2830	*****		
2831 FEDC CE FE 37		Idx #spcmag	;first part
2832 FEDF 8D 90		bsr PMSG	;of header (MDMP)
35 2833	*****		
2834 FEE1 96 48		ldaa Acnt	;get starting count
2835 FEE3 D6 47		ldab Lth	;& line length
2836 FEE5 8D 98		bsr CNTR	;the rest of the header
2837	*****		
40 2838 FEE7 D6 48		ldab Lcnt	;# of lines to dump
2839 FEE9 DE 44		Idx Msbb	;load fetch pointer
2840	*****		
2841 FEEB 8D BB	?mip:	bsr HOUTC2	;print CR, X & space
2842	*****		
2843 FEED 37	pshb		;save line counter
2844	*****		
45 2845 FEEE D6 47		ldab Lth	;this many out
2846 FEOF 8D 96		bsr PHMSG	;of this string
2847	*****		
2848 FEF2 33	pulb		;recover line counter

50

55

EP 0 549 855 A2

```

2849 FEF3 5A           decb      ;count loop down
2850 FEF4 26 F5         bne ?mp      ;on we go
2851
2852 FEF6 39           mend:   rts      ;until done # of lines
5
2853
2854           ;***** INC/DEC var addresses
2855           ;X set by RTDSRH to Adjadr
2856 FEF7
2857 FEF7 6C 00         inc 0,x    ;bump up value
2858 FEF9 20 11         bra svp
2859
10
2860 FEFB
2861 FEFB 6A 00         dec 0,x    ;bump up value
2862 FEFD 20 0D         bra svp
2863
2864 FEFF
2865 FEFF 8D FF 3D     SETVAR:  jsr HEX_BIQ   ;get a value
2866 FFO2 25 F2         bcs mend   ;no input, use old value (RTS)
2867
2868 FFO4 E6 45         lddb Lsbb   ;lower order of value input
2869 FFO6 DE 48         ldx Adjadr  ;recover value's address
2870 FFO8 E7 00         stab 0,x  ;change value
2871
20
2872           ; bsr CRLF   ;overwrite old screen data
2873 FFOA 8D 09         bsr CRR    ;overwrite old screen data
2874
2875 FFOC E6 00         svp:     lddb 0,x  ;retrieve new value
2876 FFOE 86 7C         lddb #'|' ;delimiter
2877
25
2878           ; print character in A & value in B with trailing space
2879 FF10
2880 FF10 8D 05         PCHARV:  bsr outc ;bsr OUT_QC  ;print the char (pcv)
2881
2882 FF12 17           tba      ;get value
2883           ; bra HOUTS   ;dump out and return
30
2884 FF13 8D 9A         bsr HOUTS  ;dump out
2885
2886           ; bra CRR    ;cr only & return
2887           ; fall into
2888
2889 FF15 86 0D         CRR:    lddb #CR   ;CR only
35
2890 FF17 7E FE 3E     outc:   jmp OUT_QC ;stuff & return (CRR)
2891
2892           ;*** end of PCHARV & CRR
2893
2894           ;*****
40
2895           ; PRULS| prints formated rules to screen (eventually) using adjust subs
2896           ; Redundant!
2897           ; PRULS:      ;dbg
2898           ;     ldx #S0110  ;header 1 line, 16 elements
2899           ;     jsr ENTDMP  ;do header & data @ Mabb
2900
2901           ;     ldx Adjadr
45
2902           ;     bsr HOUTC2
2903
2904           ;     ldx Mabb
2905           ;     bra HOUTC2
50
2906
2907           ;***** end of PPARS
2908           ;.page

```

EP 0 549 855 A2

```

2909      ;*****
2910      ; IN_DQ. DeQueue an input character into A
2911      ; IN_EDQ. Echoed DeQueue
2912      ; Alters A
2913      ; Use B ? indexed refs? common OUT_DQ code?
2914      ;*****
2915      ;@ use in_dqw for those callers looping with BCS to IN_DQ
2916
2917  FF1A  8D 05      IN_EDQ: bsr IN_DQ      ;get a char
2918
2919  FF1C  25 08      bcs nie      ;no input to echo so end, passing C
2920
2921      :: bsr IN_DQW      ;wait for a char
2922      :: jmp OUT_QC      ;echo it & return
2923
2924      ::IN_DQW: bsr IN_DQ      ;call it
2925      :: bcs IN_DQW      ;hang around
2926
2927      :: rts
2928
2929  FF21      IN_DQ:
2930  FF21  96 A5      ldeq in_bffil      ;get size (TST larger)
2931  FF23  26 02      bne in_cont      ;have data?
2932
2933  FF25  0D      nie: sec      ;NO, set carry & return A as zero
2934  FF26  39      nie: rts
2935
2936  FF27      in_cont:
2937  FF27  3C      pshx      ;save it now
2938  FF28  DE A8      ldx in_tpstr      ;get pointer
2939      :: ldeq 0,x      ;get char
2940  FF2A  A6 00      ldeq 0,x      ;get char
2941  FF2C
2942  FF2C  08      inx      ;skip to next spot
2943  FF2D  8C 00 BF      cpx #IN_BFEND      ;did we fall off end of buffer?
2944  FF30  23 03      bls ?in_nowrp2      ;no, skip wrap
2945
2946  FF32  CE 00 AA      ldx #IN_BUFSTA      ;yes, point back to the start
2947  FF35      ?in_nowrp2:
2948  FF35  DF A8      stx in_tpstr      ;update tail pointer
2949  FF37  7A 00 A5      dec in_bffil      ;one less occupied. Int's can't split this instruction
2950
2951      :: should ldeq bffil and compare directly?
2952      :: ldeq in_bffil      ;
2953      :: cmpe #IN_BFMIN      ;should we send XON character?
2954      :: bpl in_nxon      ;no
2955
2956      ;***** Yes, send XON, space available
2957      ;@@@ this needs development. How do you jam into output stream? Use Flags?
2958      ;in_nxon:
2959      :: tba      ;copy to a
2960  FF3A  0C      clc      ;no error flag
2961  FF3B  38      pulx      ;recover reg
2962  FF3C  39      rts      ;we're done
2963
2964      ;*****

```

50

55

## EP 0 549 855 A2

```

2965      ;HEX_BIN. Convert Ascii hex value
2966      ;DEC_BIN. Convert the characters in buffer to decimal and save in msbb/lubb
2967      ;Stops at non-numeric char which gets returned in upper case.
2968      ;Carry set if no data to do
2969      ;Mangles A,B, X
2970      ;*****
5       ;@@@ Should NOT convert char to upper case??
2972  FF3D  86 3F      HEX_BIQ:ldaa #'?' ;prompt...
2973  FF3F  BD FE 3E      jsr OUT_QC      ;for input (hexbiq)

10      HEX_BIN:
2976      bset Flag1,HFLG ;set hex bit
2977      bra ?cidd      ;skip past decimal

15      ;DEC_BIN:
2979      bcir Flag1,HFLG ;clr hex bit
2980
2981
2982      ;@@@ may want a form that skips clear?
2983  FF42  4F      ?cidd:  clra      ;clear value
2984  FF43  5F      clrb      ;the smaller way
2985  FF44  00 44      std Msbb      ;here, gets Labb too
2986      bcir Flag1,NFLG ;start out positive
2987
2988  FF46  8D 18      bsr rgchk *      ;get char & range check
2989
2990  FF48  25 34      bcs cend      ;out of bounds, return error, no input
2991
2992  FF4A  20 04      bra ?a2      ;ok, lets go with 1st time in
2993      ;bcc ?a2      ;a valid 1st char
2994      ;cmpe #'-'      ;negative sign?
2995      ;beq ?sneg      ;set negative
2996
2997      ;cmpe #'+'      ;plus
2998      ;bne cend      ;invalid 1st char
2999      ;bcir Flag1,NFLG ;clr negative flag
3000      ;bra ?a1      ;fall into additional chars loop
3001      ;?sneg:  bset Flag1,NFLG ;set negative flag
3002
3003  FF4C  8D 12      ?a1:   bsr rgchk      ;get char & check limits
3004
3005  FF4E  25 24      bcs aend      ;not valid, we're done
3006      ;bcs anend      ;not valid, we're done
35      3007  FF50
3008  FF50  80 30      ?a2:   suba #'0'      ;OK so far, strip down ascii
3009  FF52  16      tab      ;place in low half
3010  FF53  8F      xgdx      ;swap to x
3011
40      3012  FF54  DC 44      ldd Mabb      ;get old value
3013  FF56  05      asld      ;*2
3014  FF57  05      asld      ;*2 = 4
3015      ;brcir Flag1,HFLG,ddec      ;hex way?
3016
3017  FF58  05      asld      ;yes, *2 = 8
3018      ;bra ?dhex      ;do hex
45      3019
3020      ;dddec:  addd Mabb      ;+1 = 5
3021      ;?dhex:

```

50

55

EP 0 549 855 A2

3022	FF59 05	asld	; *2 = X10 or X16	
3023		xgdx	;swap back	
3024	FF5A 8F	abx	;add in digit	
3025	FF5B 3A	stx Msbb	;store it	
5	3026 FF5C DF 44	bra ?s1	;loop for more	
3027	FF5E 20 EC	*****		
3028		;Range check subr. Get character and check for numeric (or hex)		
3029		rgchk:	bsr IN_EDQ ;get char with echo	
3030	FF60 8D B8	bcs rgchk	;none avail, wait	
3031	FF62 25 FC			
10	3032	cmpl #""	;lower case?	
3033	FF64 81 60	bit ?rgck	;no, less than 1st L.C. char	
3034	FF66 2D 02	suba #320	;change to upper case	
3035		?rgck:		
3036	FF68 80 20	brclr Flag1,HFLG,dchk	;Hex accepted? Bra if not	
15	3037 FF6A	cmpl #''A'	;in range?	
3038		bit dchk	;too small, may be 0-9	
3039		cmpl #''F'		
3040	FF6A 81 41	bgt cend	;too big to be used	
3041	FF6C 2D 08	suba #7	;shift it to 10-15	
3042		bra send	;ok exit	
20	3043 FF6E 81 46	send:	clc	;return no error code & next char in upper case
3044	FF70 2E OC	rts		;we're done. Return to caller
3045		:anend:	brclr Flag1,NFLG,send	;not negative, do normal exit
3046	FF72 80 07	ldd Msbb		;get result
3047		coms		
3048		comb		
25	3049 FF74 0C	addd #1	;two's compliment negation	
3050	FF75 39	bra send	;clean exit	
3051				
3052		dchk:	cmpl #''0'	;in range?
3053		bit cend	;too small	
3054		cmpl #''9'		
3055		bgt cend	;too big, set carry	
30	3056 FF7A 81 39	ble send	;ok ending	
3057				
3058		cend:	sec	;return error code & next char in UC
3059	FF76 81 30	hend:	rts	
3060	FF78 2D 04	*****		
3061		end:	ldab #7	;starting here
35	3062 FF7A 81 39	7dvs:	jsr DERIVE	;compute the derivative data
3063				
3064	FF7C 2F F6			
3065				
3066	FF7E 0D			
3067	FF7F 39			
40	3068			
3069				
3070				
3071				
3072				
3073	FF80	DUMP:		
45	3074 FF80 BD FE 60	jsr lprompt	;fresh line & prompt with Currul & ""	
3075				
3076	FF83 C8 07			
3077	FF85 BD FB 92			
3078				

EP 0 549 855 A2

3079	FF88	5A	decb	:count down	
3080	FF89	2C FA	bge ?dvs	:until done all	
3081					
3082	FF88	CE 00 60	Idx #Anidat	:Data stored here every OC1 int	
5	3083	FF8E	CE 10	Idx #16 :length of string	
	3084	FF90	BD FE 88	jsr PHMSG :print in hex (dump)	
	3085				
	3086	FF93	B6 10 03	Idaa PORTC :current digital values	
	3087	FF96	8D 3A	ber outb ;jsr HOUTS ;out + spc (dump)	
	3088				
10	3089			:using RTC for delay	
	3090	FF98	C6 04	Idx #SDRATE :get rate value	
	3091	FF9A	CE 10 00	Idx #REG :get the base	
	3092	FF90	1D 25 BF	?wrtc: bclr _TFLG2,x,\$BF ;R/M/W knock down RTIF (\$40)	
	3093	FFA0		?ckip: jsr IN_DQ ;any keys? (dump)	
	3094	FFA0	BD FF 21		
	3095				
15	3096	FFA3	24 09	bcc ?srt ;yes, do em	
	3097				
	3098	FFA5	1F 25 40 F7	bclr _TFLG2,x,\$40,?ckip ;hang around until RTIF high	
	3099				
20	3100	FFA9	5A	decb	:count out time
	3101	FFAA	2E F1	bgt ?wrtc	:hang around some more
	3102				
	3103	FFAC	20 D2	bra DUMP	:another time around
	3104				
	3105			?srt: jsr RTDSPH ;check for escape or tune	
	3106	FFAE	81 1B	?srt: cmpa #ESC ;escape	
	3107	FFB0	26 CE	bne DUMP	
25	3108				
	3109	FFB2	39	dend: rts	
	3110				
	3111			;*****	
	3112			:FLAGS  toggles various flag bits	
30	3113	FFB3	8D 88	FLAGS: ber HEX_BIQ	:get bit #
	3114	FFB5	25 FB	bcs dend	:no value, skip out
	3115				
	3116	FFB7	CE 00 40	Idx #Flag1	:start here
	3117			; lddb Lsbb	:get value
	3118			; cmpb #16	:upper limit
	3119			; bhi dend	;too big (this covers negative too since unsigned)
35	3120			; bmi dend	:negative, so skip
	3121				
	3122			; cmpb #8	:threshold for Flag2
	3123			; bit ?flg1	;use as is
	3124				
	3125			; inx	:point to Flag2
40	3126			; cmpb #16	:threshold for Flag3
	3127			; bit ?flg1	;ok
	3128				
	3129			; inx	:point to Flag3
	3130			?flg1:	
	3131			;18 ^ vs 12 v	
45	3132	FFBA	96 45	Idaa Lsbb	:get value
	3133	FFBC	16	tab	:copy
	3134	FFBD	56	rorb	:get highest 5 bits in position
	3135	FFBE	56	rorb	

50

55

EP 0 549 855 A2

	3136	FFBF	56	rrb	
	3137	FFC0	C1 04		cmpb #4 ;upper limit with 4 flags
	3138	FFC2	22 EE		bhi dend ;too big so skip out
	3139				
5	3140	FFC4	3A		abx ;index into Flags table
	3141				
	3142	FFC5	16	tab	;copy again
	3143	FFC6	C4 07		andb #37 ;mask to remainder modulo 8
	3144	FFC8	4F	cira	;start empty
	3145	FFC9	0D		sec ;make a high bit
	3146				
10	3147	FFCA	49	?aip:	rola ;shift across
	3148	FFCB	5A		decb ;count it
	3149	FFCC	2C FC		bge ?aip ;until -1 (base 0)
	3150				
	3151	FFCE	A8 00		eora 0,x ;flip the bits into a
	3152	FFD0	A7 00		staa 0,x ;and put back
15	3153	FFD2	7E FE AF	outb:	jmp HOUTS ;echo result & return (prompt)
	3154				
	3155				
	3156				;*****
	3157				;TEST1; tests whatever
	3158				;*****
20	3159				;TEST1:
	3160			:	inc SPDR ;bump up spi data
	3161			:	ldy #100 ;preload this value
	3162			:	jsr wt10 ;wait a bit
	3163				
	3164			:	Idx Flag1 ;getem both (DATA)
25	3165			:	jsr HOUTC2 ;dumpem
	3166				
	3167			:	Idx #SPCR ;SPI
	3168			:	ldab #3 ;length, prints all three SPI regs
	3169			:	jmp PHMSG ;message
	3170	FFD5			
30	3171			:	ldaa Anddat ;get ch0 value
	3172			:	jmp LD_MOTOR ;use it here
	3173				
	3174			:	jsr HEX_BIO ;get a value
	3175			:	ldaa Msbb
	3176			:	ber HOUTS ;upper
	3177			:	ldaa Lsbb
35	3178			:	jmp HOUT ;print it and return
	3179				.page
40					
	3180				;*****
	3181				;HELP; print out screen of help info
	3182				;*****
	3183			:	;HELP: Idx #HLPMSG ;Get pointer
	3184			:	jmp PMSG ;Print help string & return
	3185				
	3186				;HLPMSG: db CR,LF
45	3187			:	db "? Adj Dmp Entr Mdmp Tun Suv "Rat ~Tst",CR,LF + \$80
	3188				.page

50

55

```

3189      ;*****
3190      ;Interrupt Vectors
3191      ;*****
5       3192  FFD5      ABSOLUTE      ;absolute positions
3193      ;      org $FFD2      ;reserved but we'll use the space
3194      ;RSTT:  jmp START    ;unknow ints trap to here and cause restart
3195      ;*****
3196  FFD6      org $FFD6
3197  FFD6  FDCB      dw SCIINT    ;FFD6 SCI int
3198  FFD8  FBC5      dw START     ;FFD8 SPI int
10      3199  FFDA  FBC5      dw START     ;FFDA PA1
3200  FFDC  FBC5      dw START     ;FFDC PAOV
3201  FFDE  FBC5      dw START     ;FFDE TO1
3202
3203      ;      org $FFE0
15      3204  FFE0  FBC5      dw START     ;FFE0 OC5 int
3205  FFE2  FBC5      dw START     ;FFE2 OC4
3206  FFE4  FBC5      dw START     ;FFE4 OC3
3207  FFE8  FBC5      dw START     ;FFE8 OC2
3208  FFE8  F947      dw TIMINT    ;FFE8 OC1
3209  FFEA  FBC5      dw START     ;FFEA IC3
3210  FFEC  FBC5      dw START     ;FFEC IC2
20      3211  FFEE  FBC5      dw START     ;FFEE IC1
3212
3213      ;      org $FFF0
3214  FFF0  FBC5      dw START     ;FFF0 Real Time Int
3215  FFF2  FBC5      dw START     ;FFF2 Interrupt ReQuest
3216  FFF4  FE36      dw XRET      ;FFF4 XIRQ quick RTI
25      3217  FFF8  FBD6      dw ISTART    ;FFF8 SoftWare Int
3218  FFF8  FBD6      dw ISTART    ;FFF8 Illegal INSTRUCTION
3219  FFFA  FBC5      dw START     ;FFFA Computer Operating Property watchdog
3220  FFFC  FBC5      dw START     ;FFFC CLOCK Monitor
3221  FFFE  FBC5      dw START     ;FFFE power on RESET
3222
30      3223  0000      END        ;of real code

```

Lines Assembled : 3223      Assembly Errors: 0

35

## Claims

1. In an above knee prosthesis (AKP) having upper and lower leg segments and a connecting knee joint, the improvement comprising:  
 a linear, hydraulic damper for separately and variably damping each of flexion and extension rotational movements of the knee joint;  
 electronic sensing means for measuring each of AKP knee angle and lower leg segment strain and emitting signals indicative thereof;  
 actuating means for adjusting the damper to vary damping of the knee joint in flexion and extension; and  
 programmed computer means for receiving the emitted signals from the sensing means and comparing them to stored threshold values which are indicative of pre-determined transition points selected for adjustment of at least one of flexion and extension damping, and, when the received signal values correlate with stored values, causing the actuating means to vary damping.
2. A method for controlling the knee joint of an above knee prosthesis having a knee joint, lower leg and ankle, comprising:  
 storing, in a computer memory, threshold values of lower leg strain and knee angle, which values are indicative of the knee joint bending in stance phase, of anterior positioning of the center of gravity of body weight relative to the ankle, and of swing phase, all in the course of a step along a level surface;  
 continuously sensing lower leg strain and knee angle during use of the prosthesis and producing

electronic signals corresponding thereto;

comparing the signals against the stored threshold values and, when the signals substantially correlate with threshold values, automatically altering the rate of rotation of the knee joint in one or both of flexion and extension, as required.

- 5       3. A bi-directional variable linear hydraulic damper for use in an above knee prosthesis, comprising:  
a hollow closed cylinder comprising end walls and a side wall forming a chamber for retaining  
hydraulic fluid, each end wall forming a rod opening;  
a cylindrical hollow piston disposed in the cylinder chamber and adapted to slide longitudinally  
therein, said piston having axial rods extending through the rod openings in sealed engagement with  
the cylinder;  
said piston carrying an exterior circumferential seal ring between its ends, said seal ring being in  
sealing relationship with the cylinder side wall, said piston being formed by end walls and a side wall,  
said piston forming a first aperture through its wall above the seal ring and a second aperture through  
its wall below the seal ring;  
a first one way check valve controlling the first aperture for enabling ingress of fluid into the piston  
chamber from the first end of the cylinder chamber;  
a second one way check valve controlling the second aperture for enabling ingress of fluid into the  
piston chamber from the second end of the cylinder chamber;  
a first pair of diametrically opposed ports extending through the piston side wall adjacent its first  
end, on one side of the seal ring;  
a second pair of diametrically opposed ports extending through the piston side wall adjacent its  
second end, on the other side of the seal ring; and  
valve means for progressively reducing or increasing the effective area available for fluid flow of the  
first ports and separately progressively reducing or increasing the effective area available for fluid flow  
of the second ports.

10     4. In an above knee prosthesis (AKP) for use by a human user, said AKP having upper and lower leg  
segments, a knee joint connecting the segments, and a foot attached to the base of the lower leg  
segment, the improvement comprising:  
means, pivotally connected with the leg segments, for separately and variably damping each of  
flexion and extension rotational movements of the knee joint;  
electronic sensing means for monitoring AKP knee angle and position of the center of gravity of the  
user's body relative to the AKP foot and emitting signals indicative thereof;  
actuating means for adjusting the damping means to vary damping of the knee joint; and  
programmed computer means for receiving the emitted signals from the sensing means and  
continuously establishing from said signals the state of the AKP in the course of a movement and  
activating the actuating means to vary damping to substantially simulate natural knee action.

15     5. The improvement as set forth in claim 4 wherein the damping means comprises:  
a pair of closed chambers for containing hydraulic fluid,  
means, connected to the leg segments and forming two passageways connecting the chambers,  
for moving fluid from one chamber to the other through one of the passageways when the leg  
segments are moving together and through the other of the passageways when the leg segments are  
moving apart, and  
means for regulating the flow of fluid through each passageway;  
said actuating means being adapted to adjust the regulating means to vary damping of the knee  
joint.

20     6. The improvement as set forth in claim 4 wherein the damping means is a bi-directional variable linear  
hydraulic damper comprising:  
a hollow closed cylinder comprising end walls and a side wall forming a chamber for retaining  
hydraulic fluid;  
a cylindrical hollow piston disposed in the cylinder chamber and adapted to slide longitudinally  
therein;  
said piston carrying an exterior circumferential seal ring between its ends, said seal ring being in  
sealing relationship with the cylinder side wall, said piston being formed by end walls and a side wall,  
said piston forming a first aperture through its wall above the seal ring and a second aperture through

its wall below the seal ring, said piston dividing the cylinder chamber into closed first and second end chambers;

5 a first one way check valve controlling the first aperture for enabling ingress of fluid into the piston chamber from the first end chamber;

a second one way check valve controlling the second aperture for enabling ingress of fluid into the piston chamber from the second end chamber;

10 a first pair of diametrically opposed ports extending through the piston side wall adjacent its first end, on one side of the seal ring;

a second pair of diametrically opposed ports extending through the piston side wall adjacent its second end, on the other side of the seal ring; and

15 valve means for progressively reducing or increasing the effective area available for fluid flow of the first ports and separately progressively reducing or increasing the effective area available for fluid flow of the second ports;

said actuating means being adapted to adjust the valve means to vary damping of the knee joint.

- 20 7. The improvement as set forth in claim 4 wherein the programmed computer means is adapted to compare the emitted signals against stored threshold values indicative of transition points between states of a repetitive movement of the AKP and, when the signals substantially correlate with threshold values, to alter the rate of rotation of the knee joint in one of or both of flexion and extension.
- 25 8. The improvement as set forth in claim 7 wherein the stored threshold values are selected from the group consisting of the absolute and derivative values of knee angle and the position of the center of gravity of the user's body relative to the AKP foot, the duration from the last transition point and the possible future states in the course of the movement.
- 30 9. The improvement as set forth in claim 8 wherein:  
the sensing means for monitoring the position of the center of gravity of the user's body relative to the AKP foot consists of means for monitoring lower leg strain.
- 35 10. The improvement as set forth in claim 6 wherein the programmed computer means is adapted to compare the emitted signals against stored threshold values indicative of transition points between states of a repetitive movement of the AKP and, when the signals substantially correlate with threshold values, to alter the rate of rotation of the knee joint in one of or both of flexion and extension.
- 40 11. The improvement as set forth in claim 10 wherein the stored threshold values are selected from the group consisting of the absolute and derivative values of knee angle and the position of the center of gravity of the user's body relative to the AKP foot, the duration from the last transition point and the possible future states in the course of the movement.
- 45 12. The improvement as set forth in claim 11 wherein:  
the sensing means for monitoring the position of the center of gravity of the user's body relative to the AKP foot consists of means for monitoring lower leg strain.

50

55

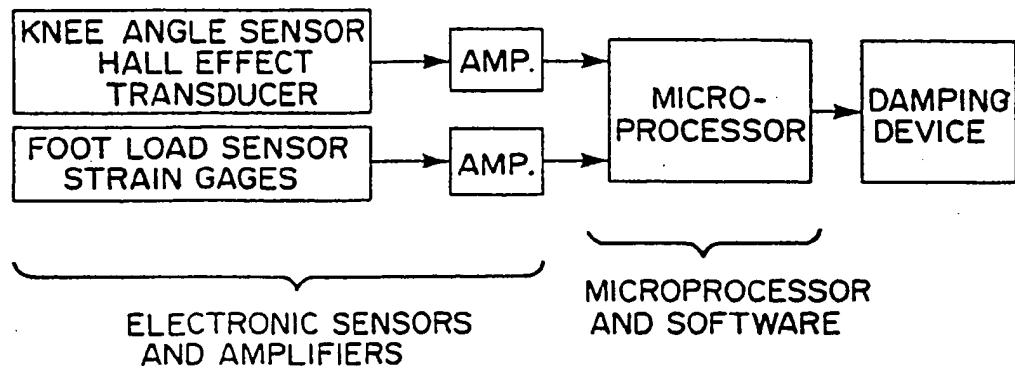


Fig. 1.

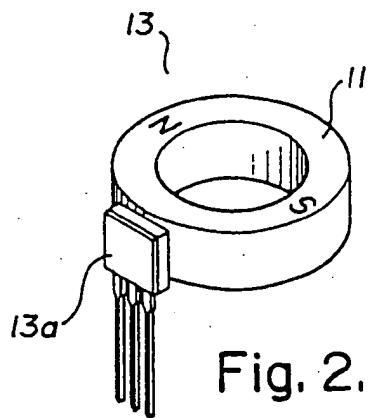
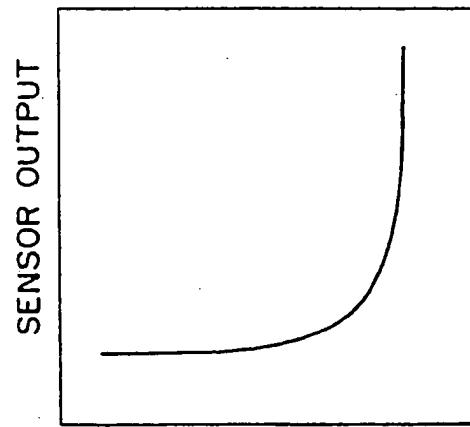


Fig. 2.



KNEE ANGLE

Fig. 3.

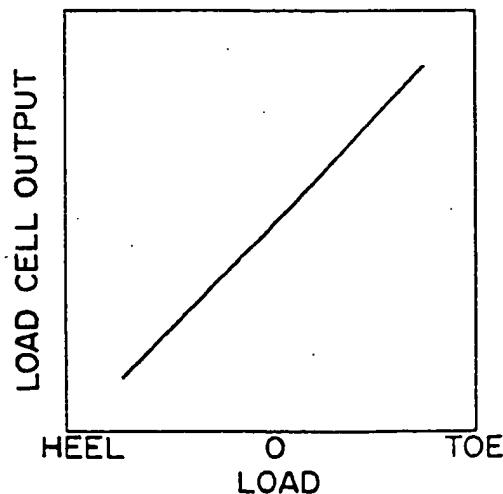


Fig. 5.

Fig. 4.

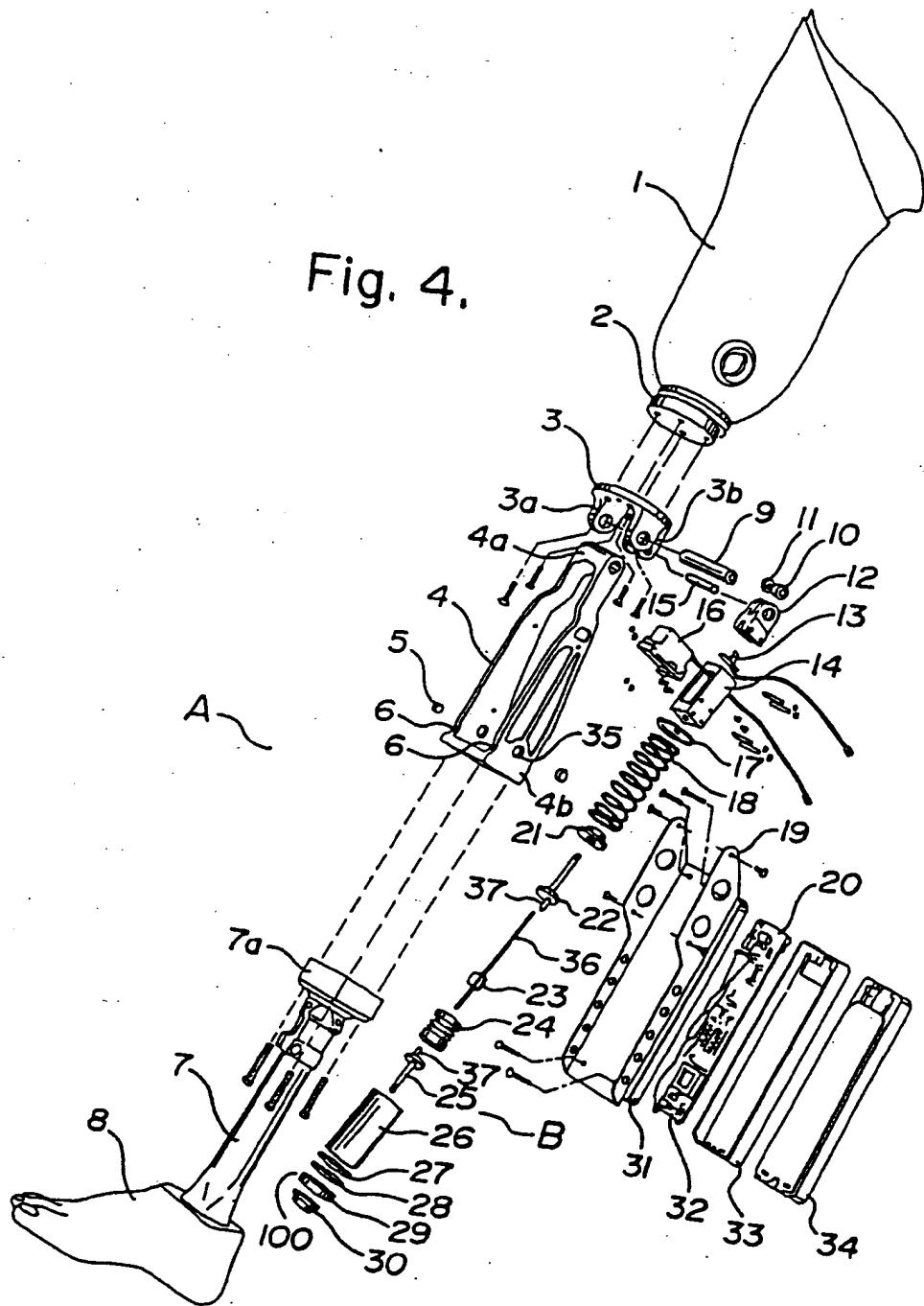


Fig. 4a.

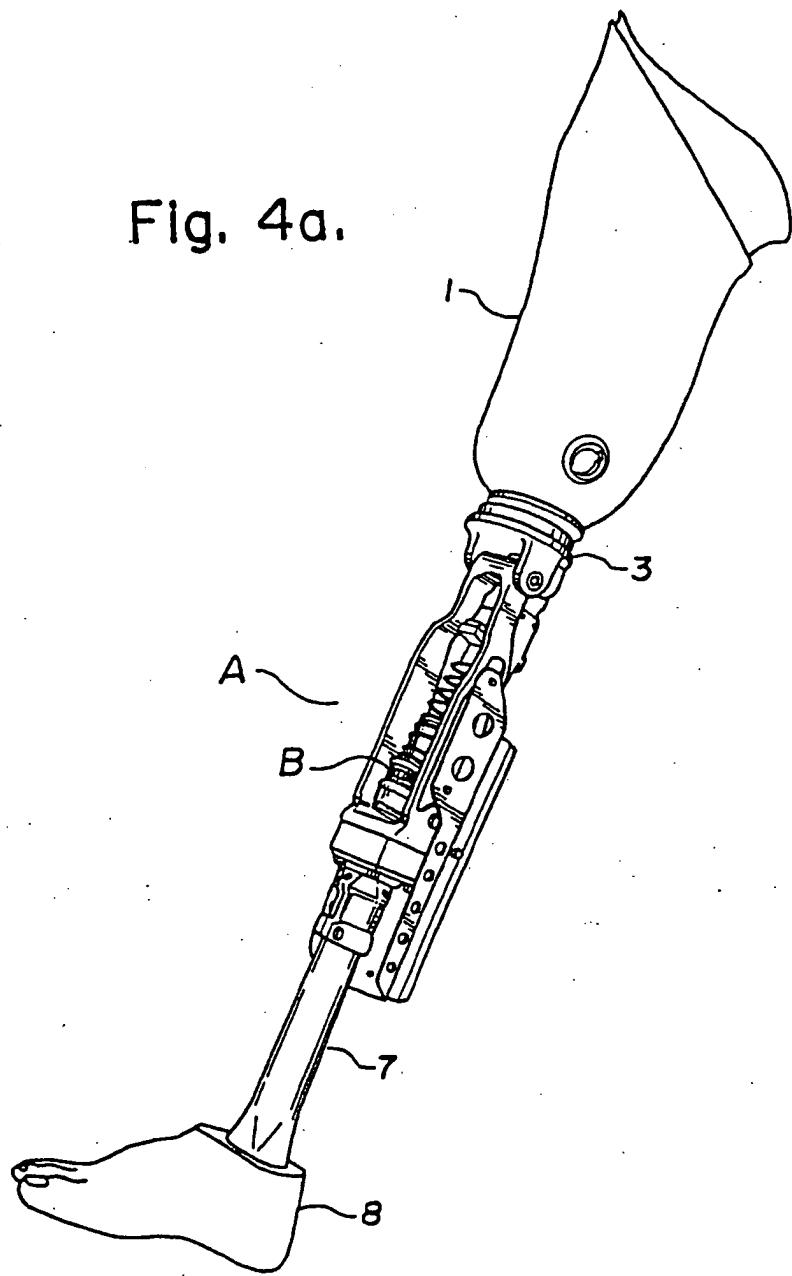
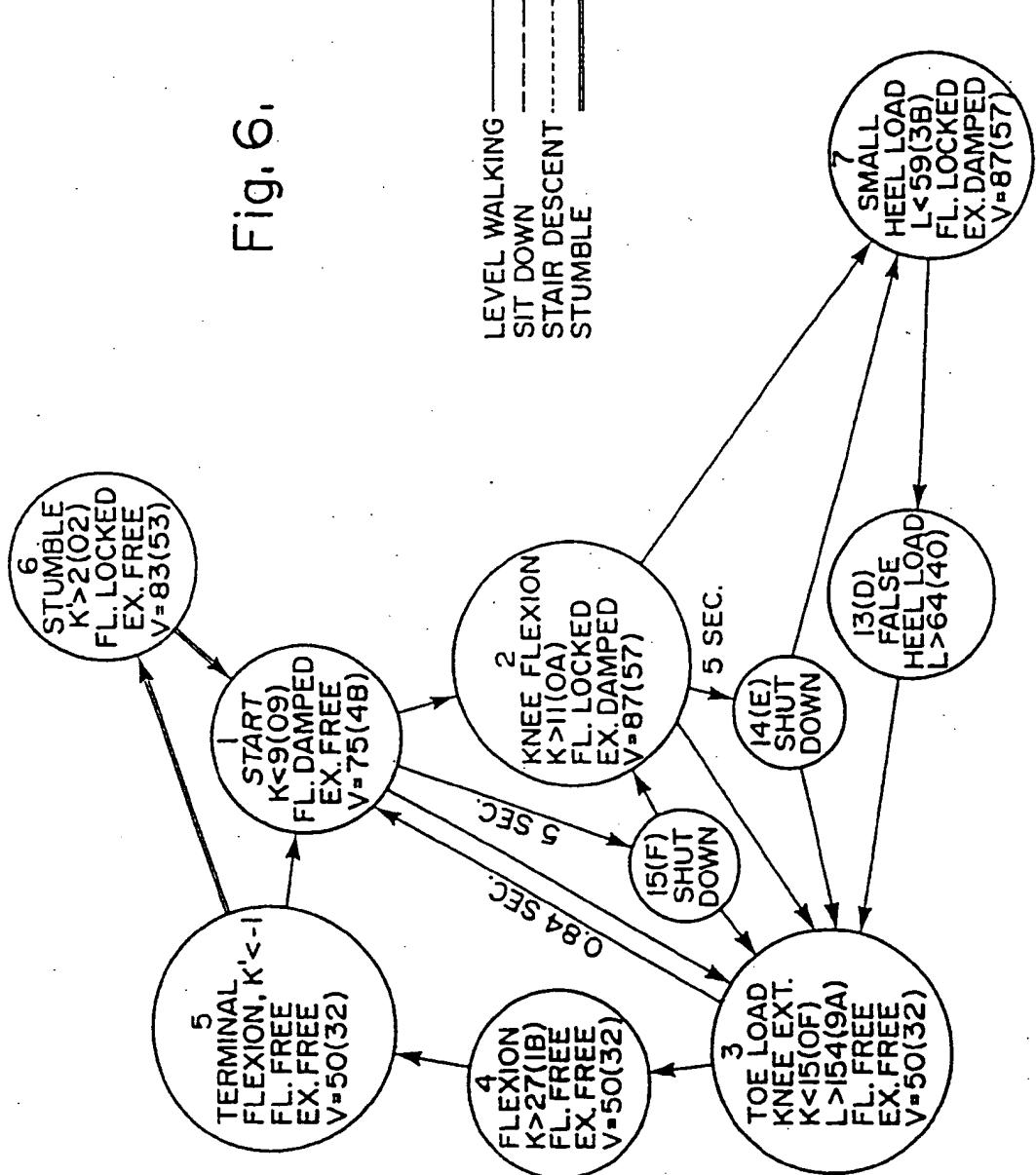
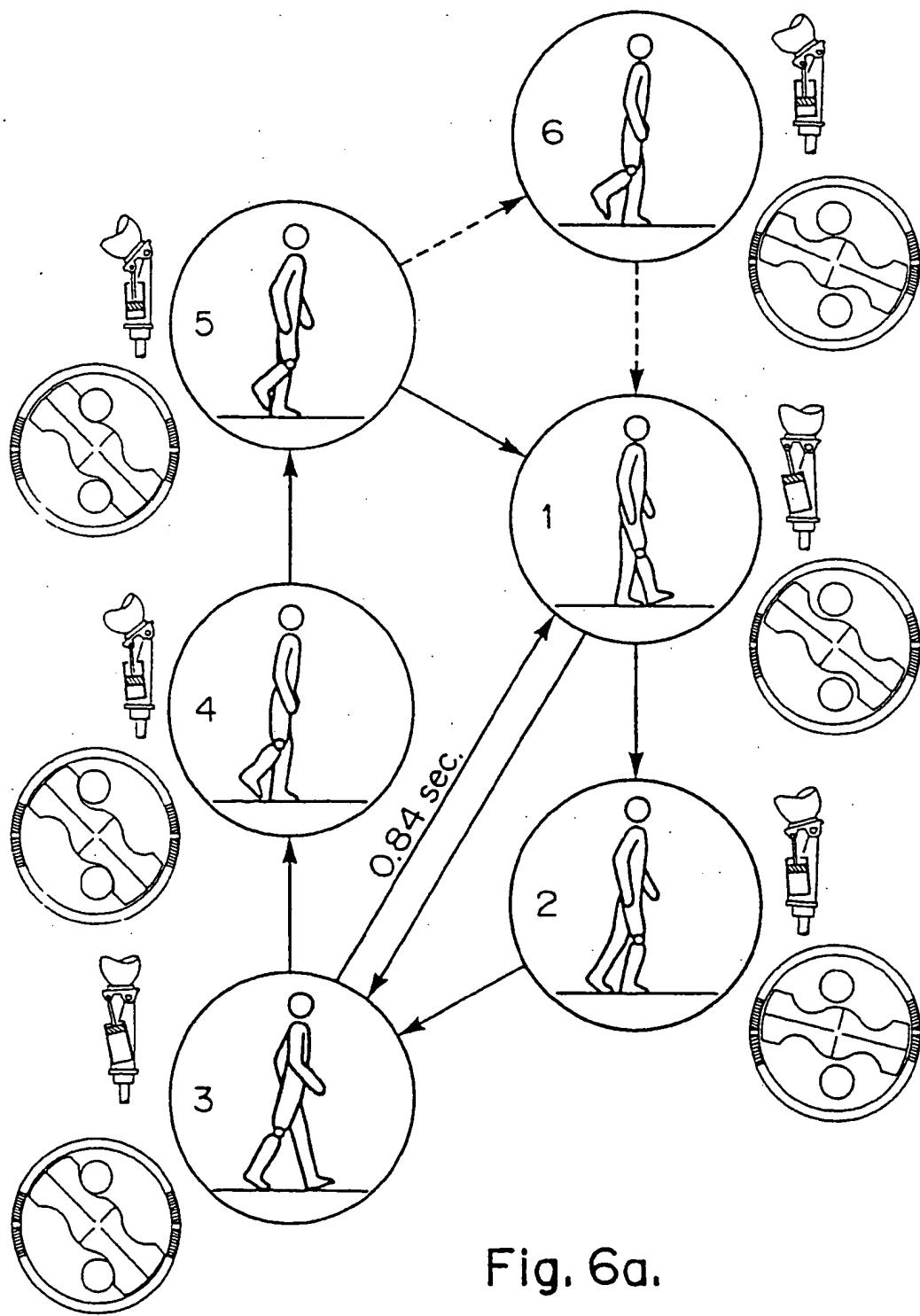


Fig. 6.





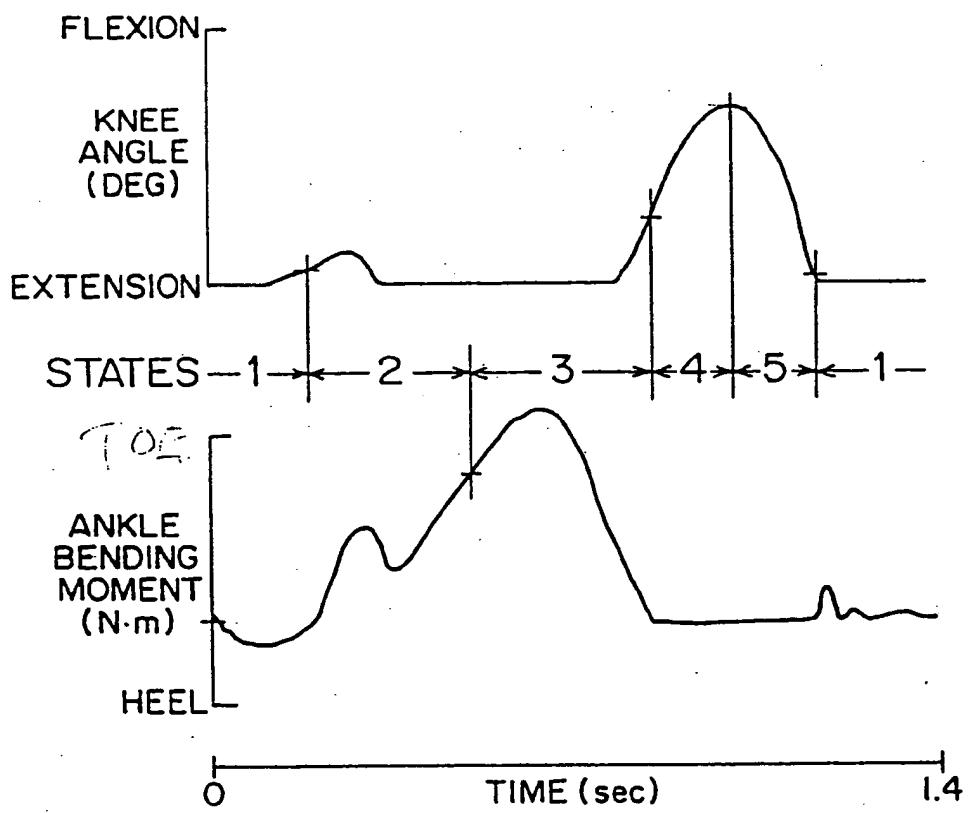


Fig. 7.

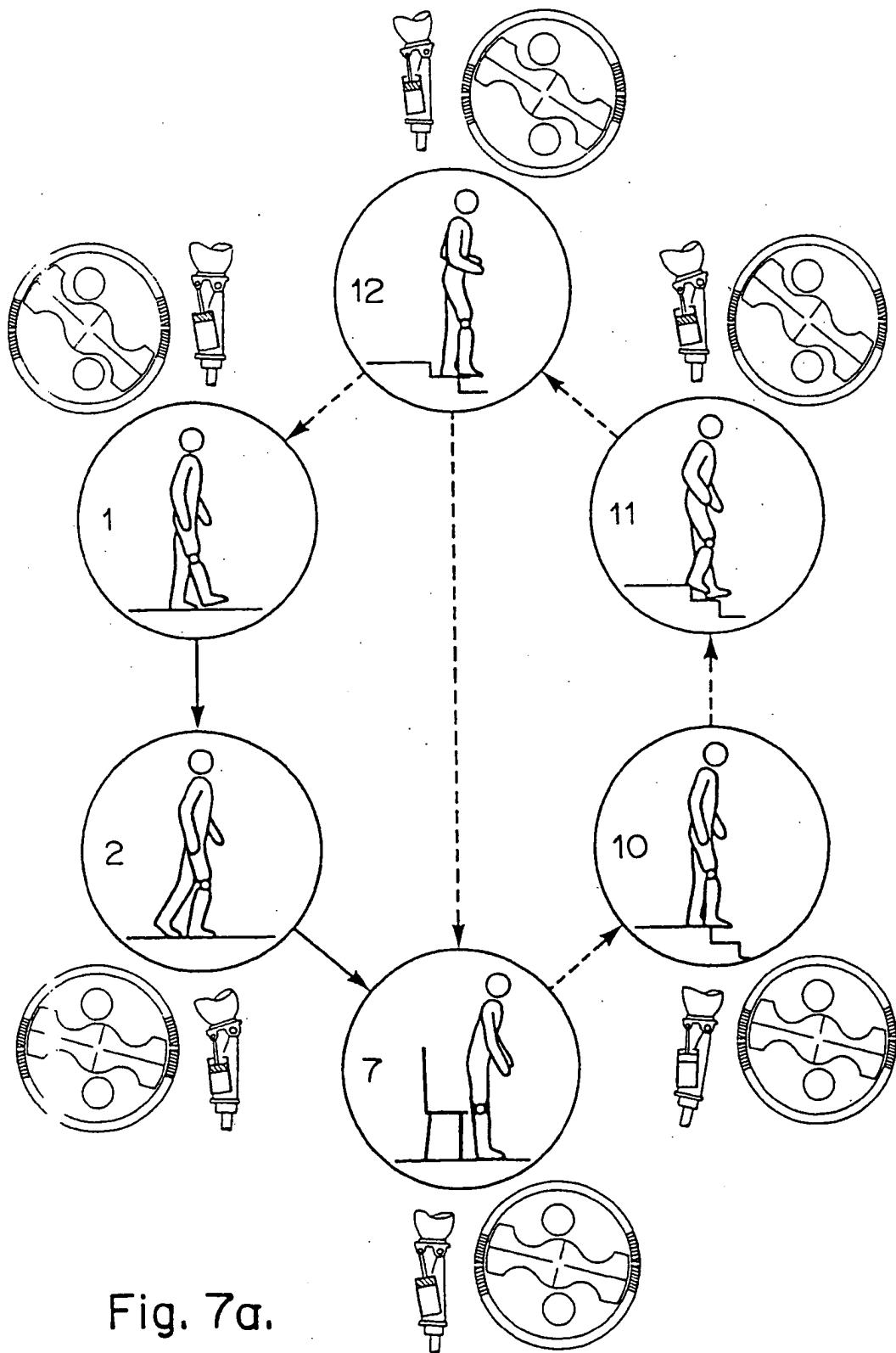
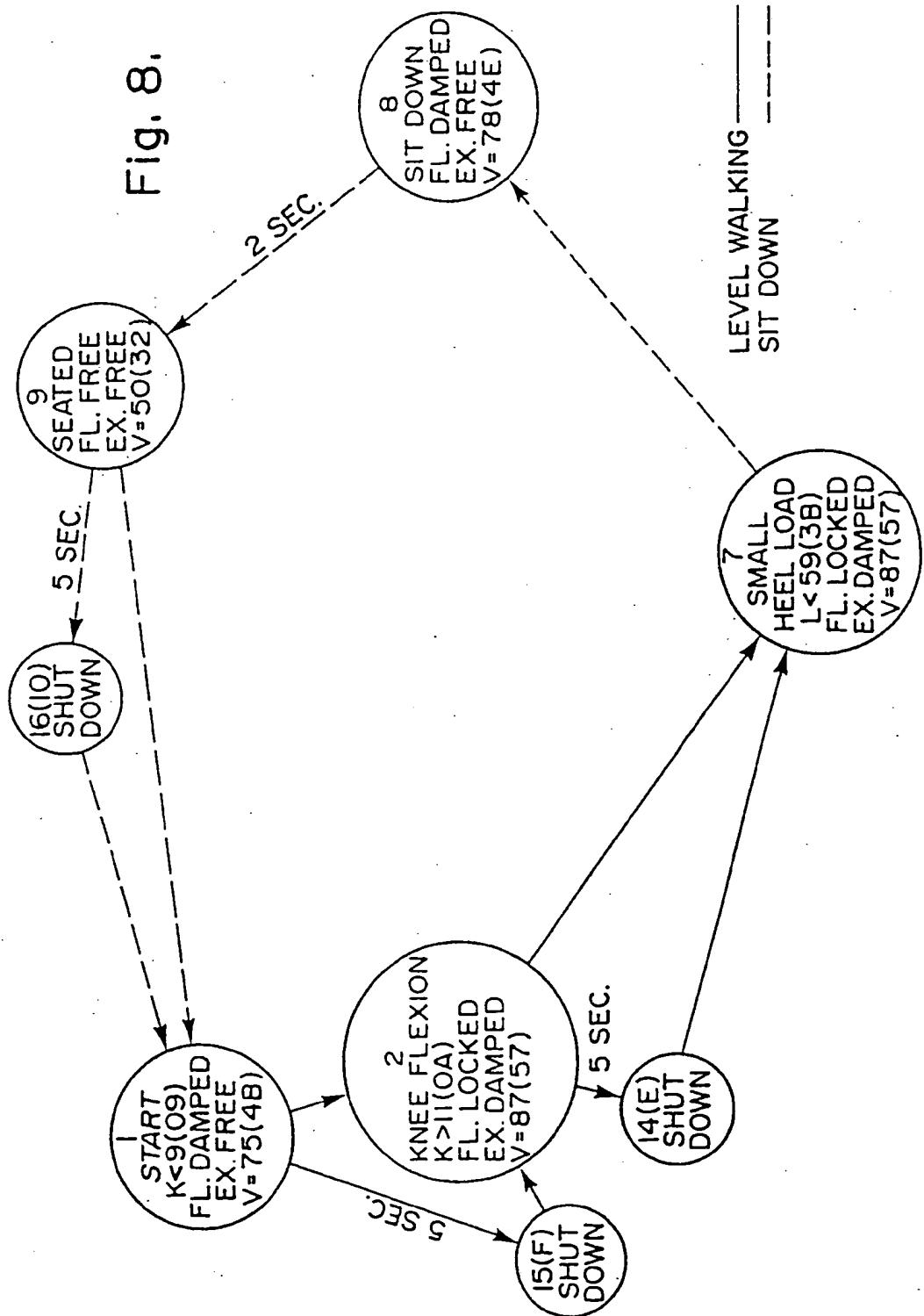


Fig. 7a.

୪୮



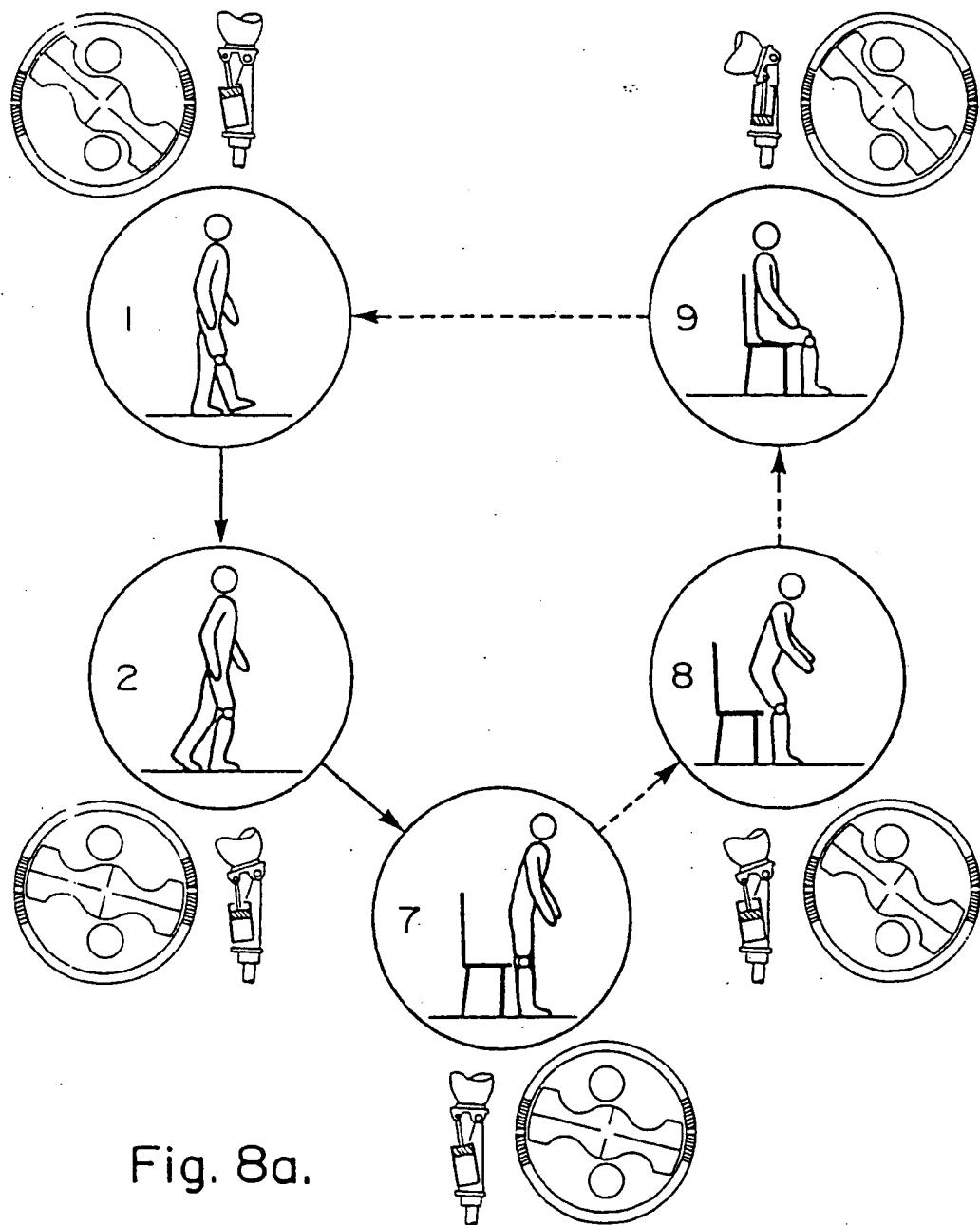


Fig. 8a.

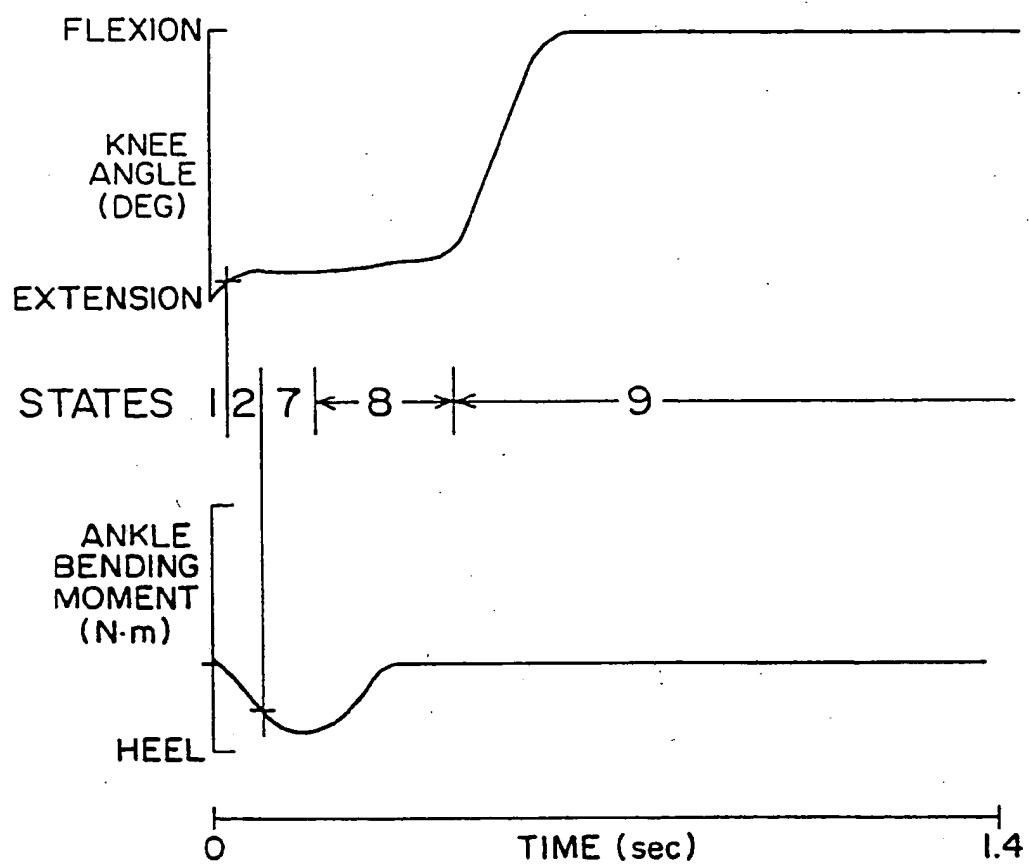
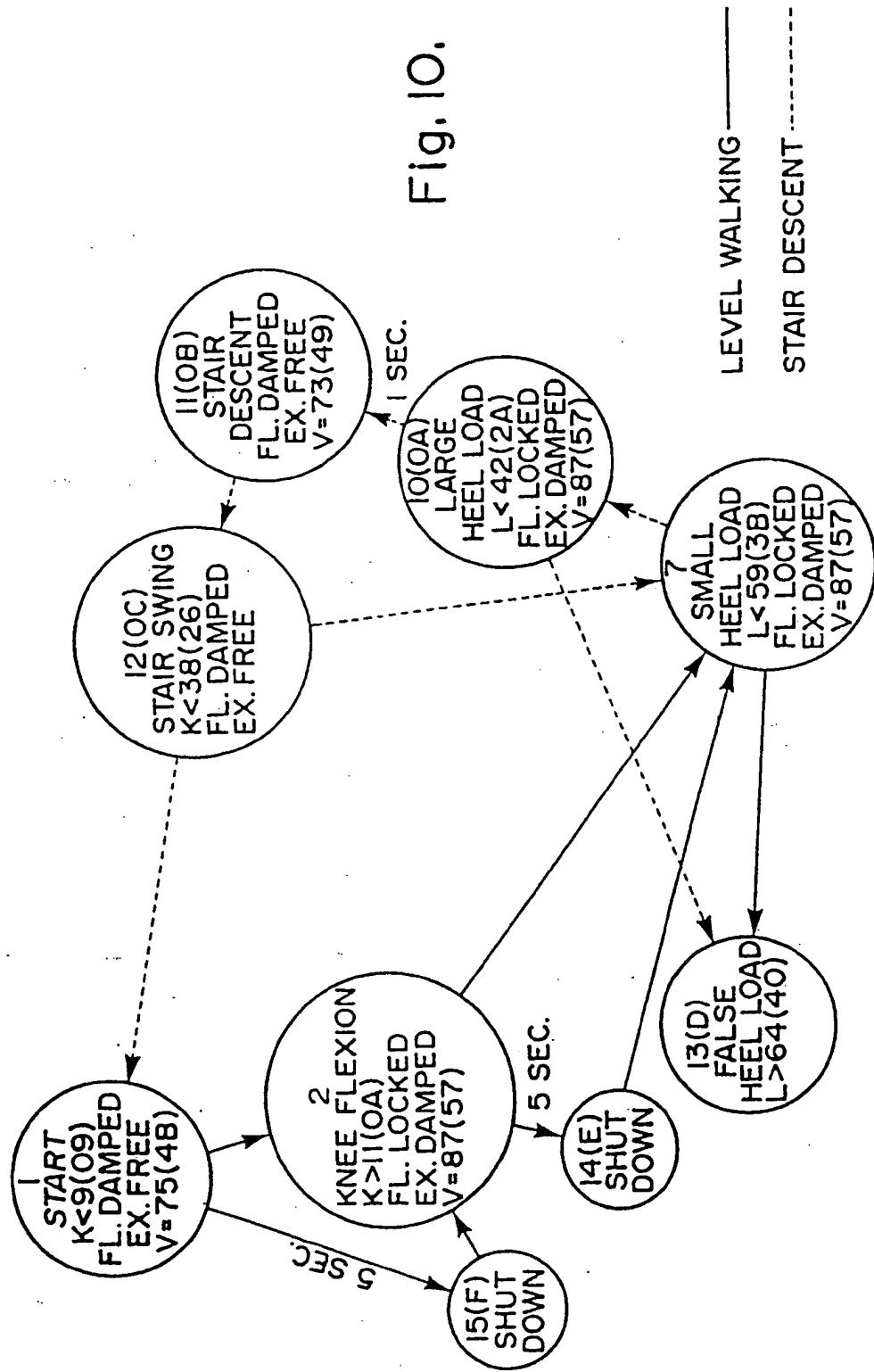


Fig. 9.

Fig. 10.



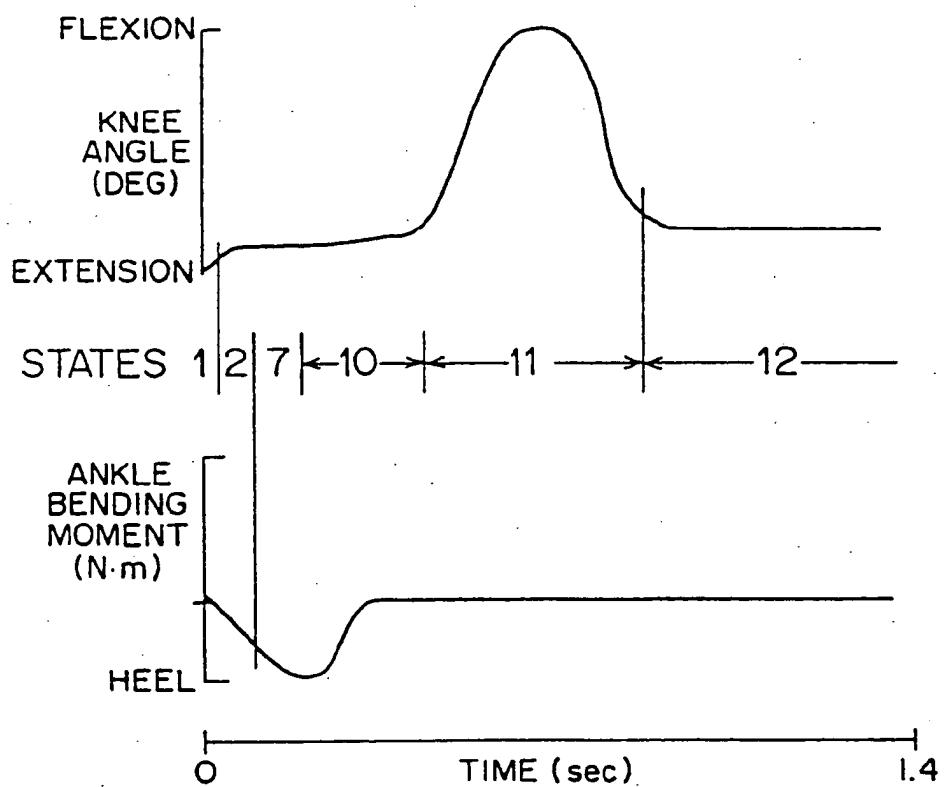
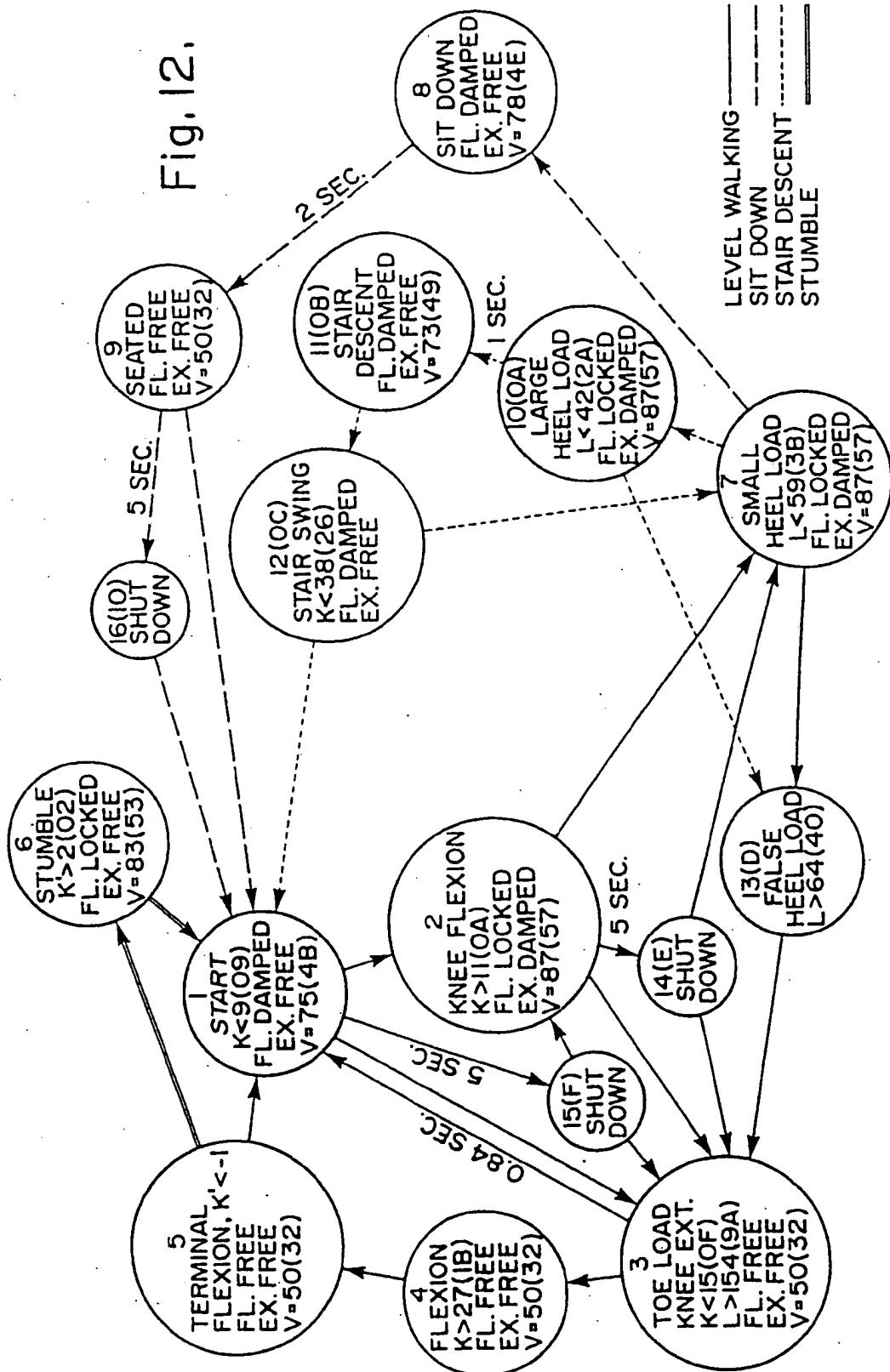
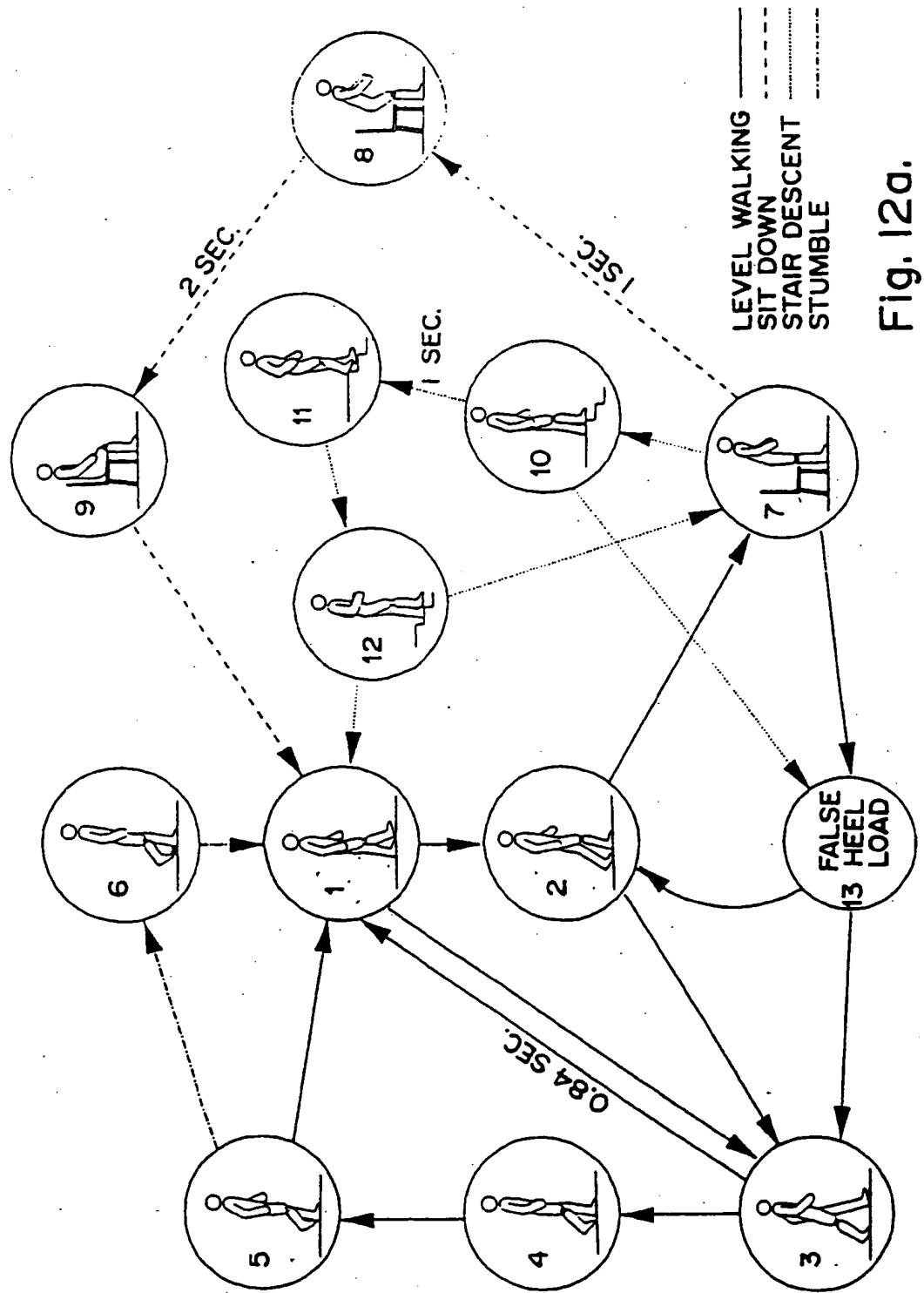


Fig. 11.

Fig. 12.





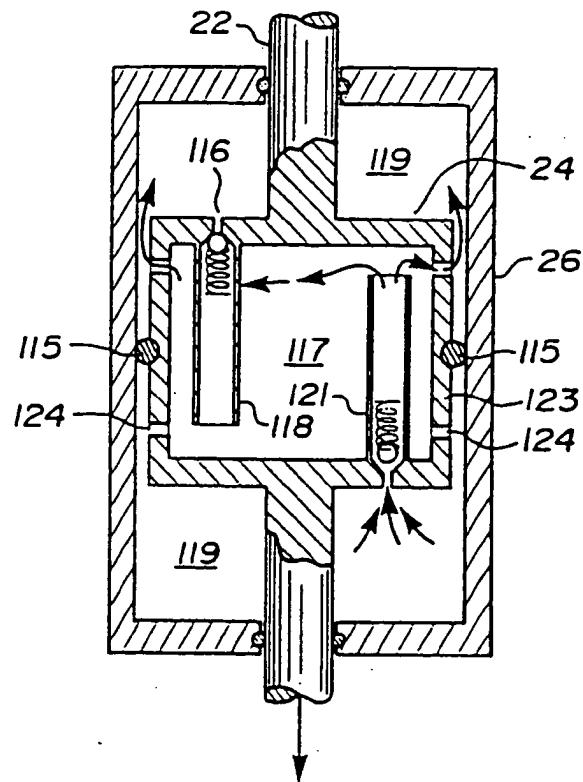


Fig. 13.

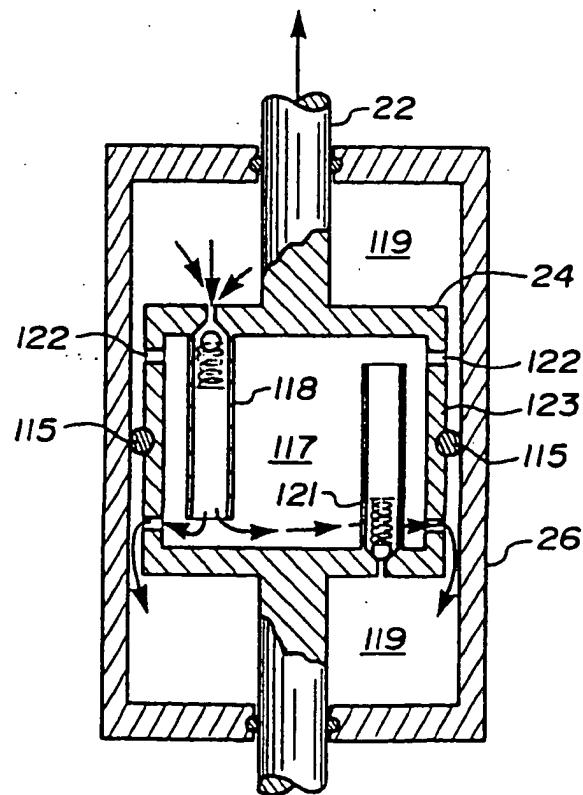


Fig. 14.

Fig. 15.

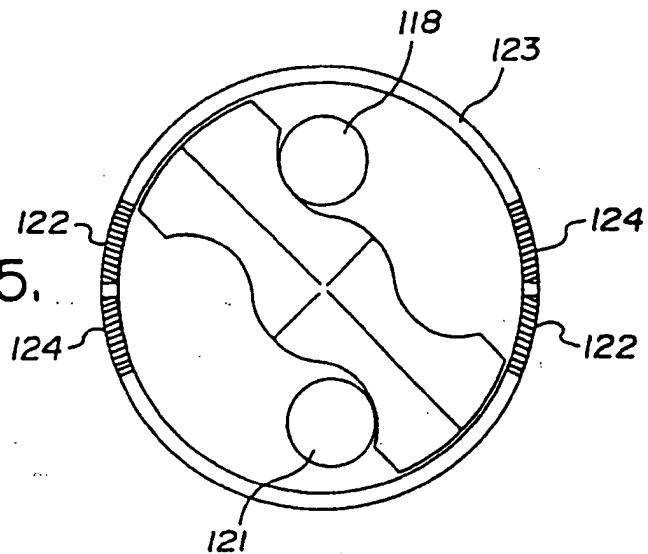


Fig. 16.

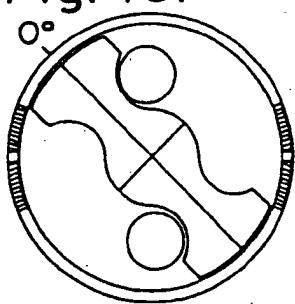


Fig. 17

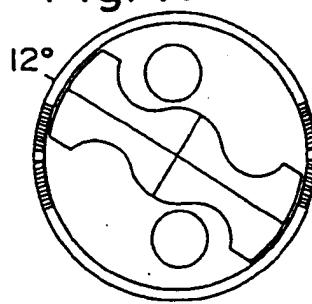


Fig. 18.

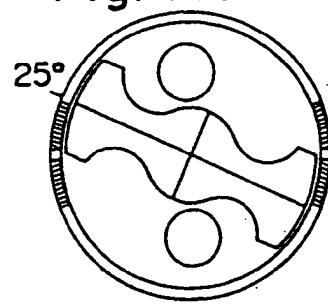


Fig. 19.

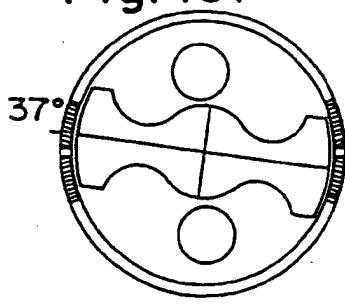


Fig. 20.

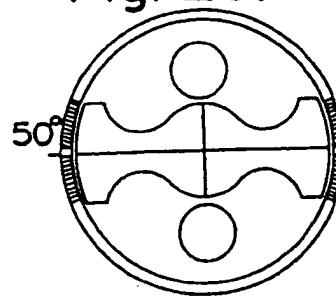


Fig. 21.

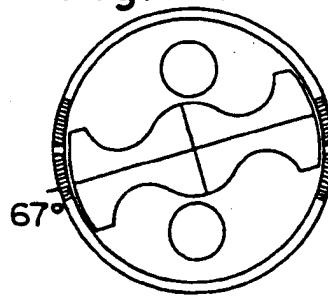


Fig. 22.

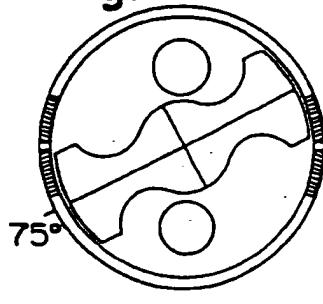


Fig. 23.

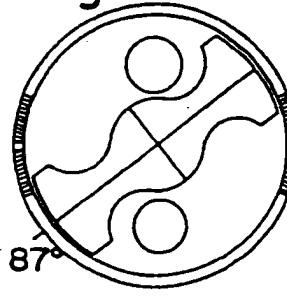
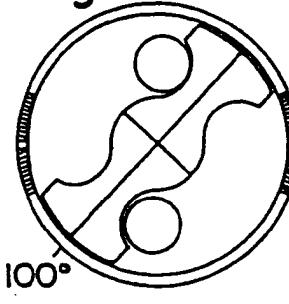


Fig. 24.



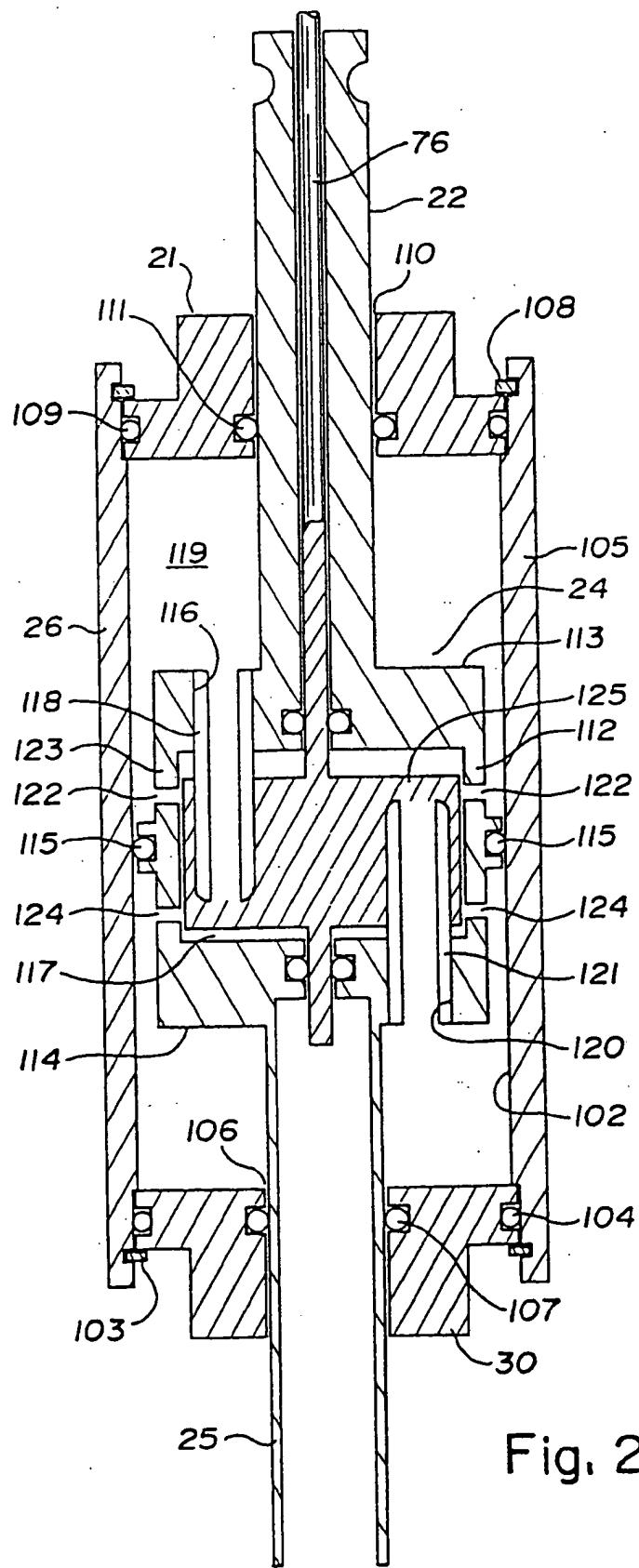


Fig. 25.

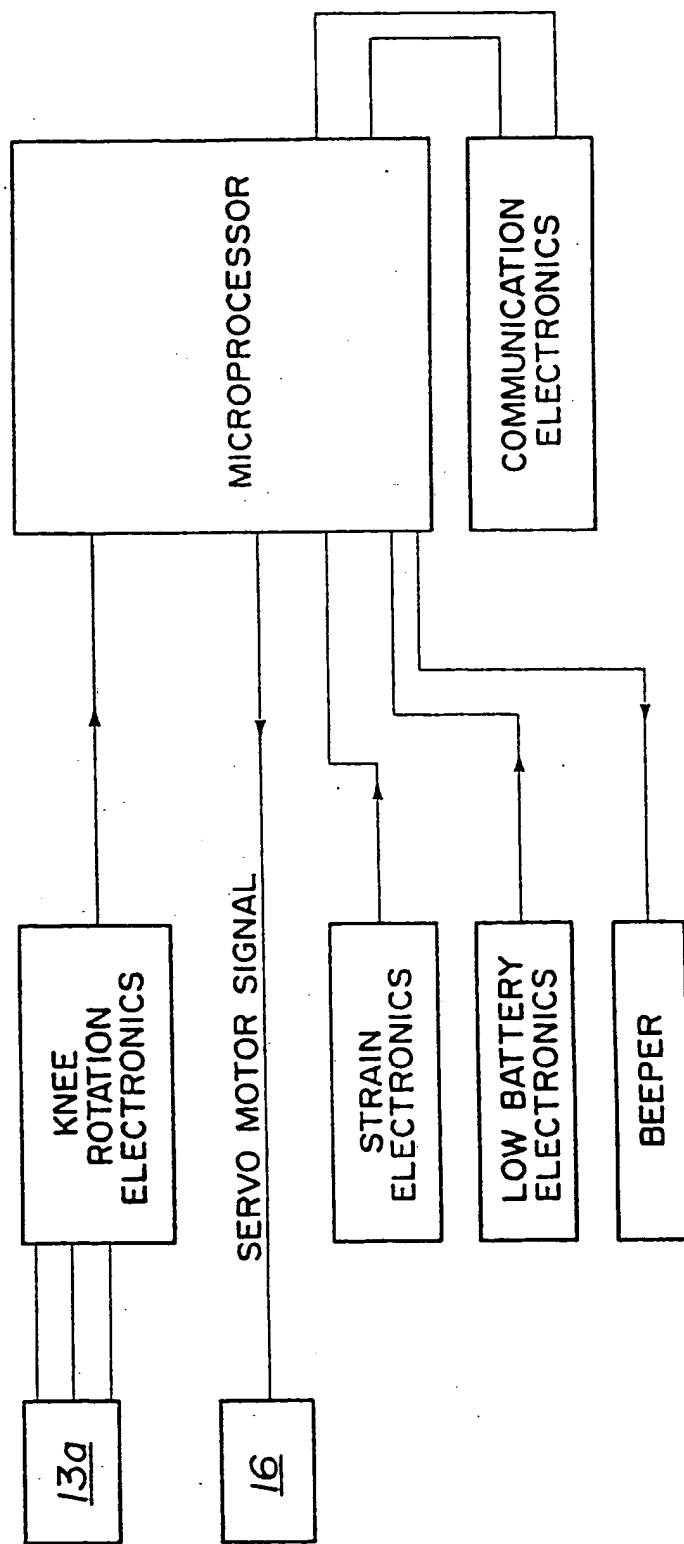


Fig. 26.

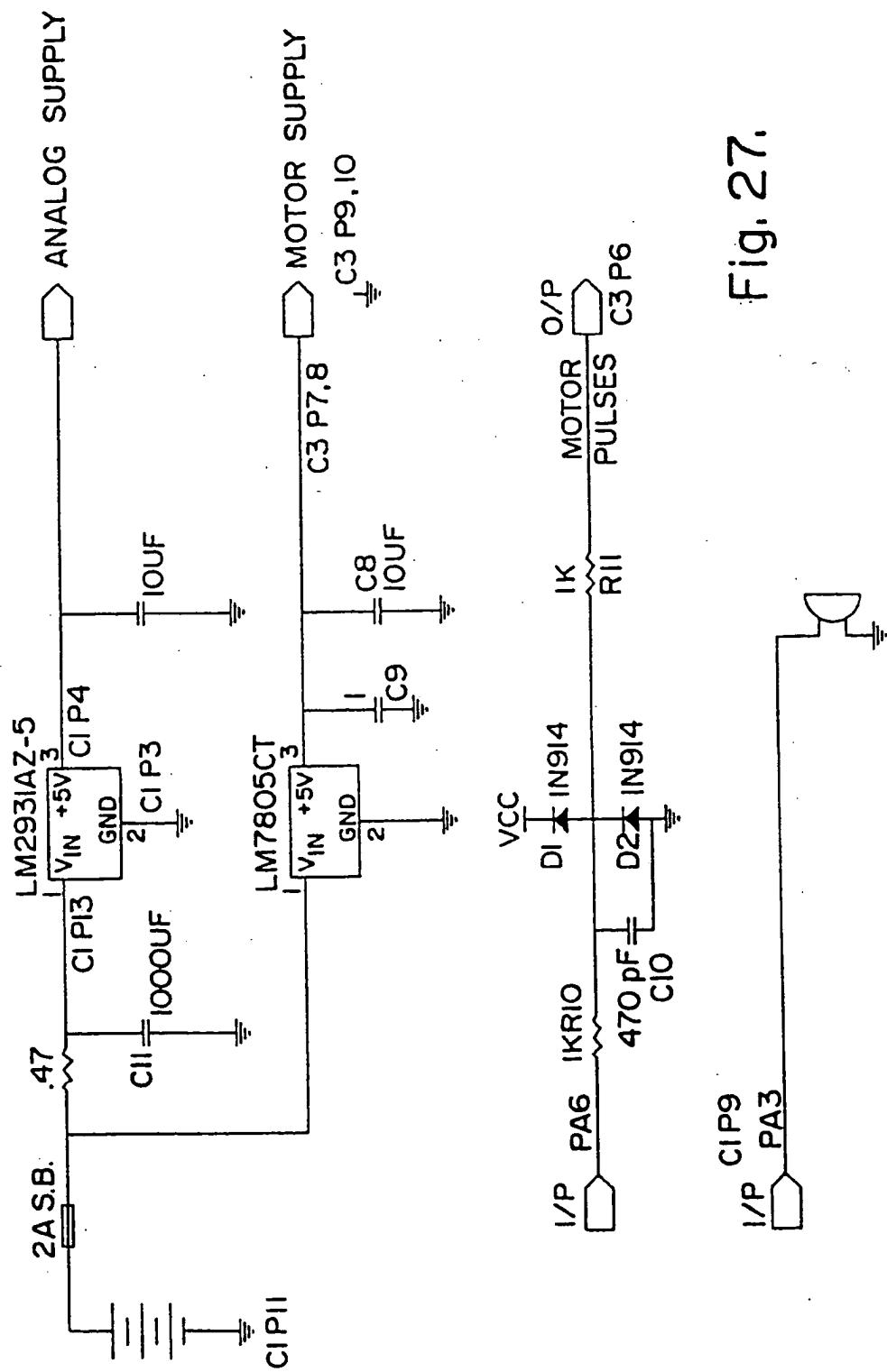


Fig. 27.

Fig. 28.

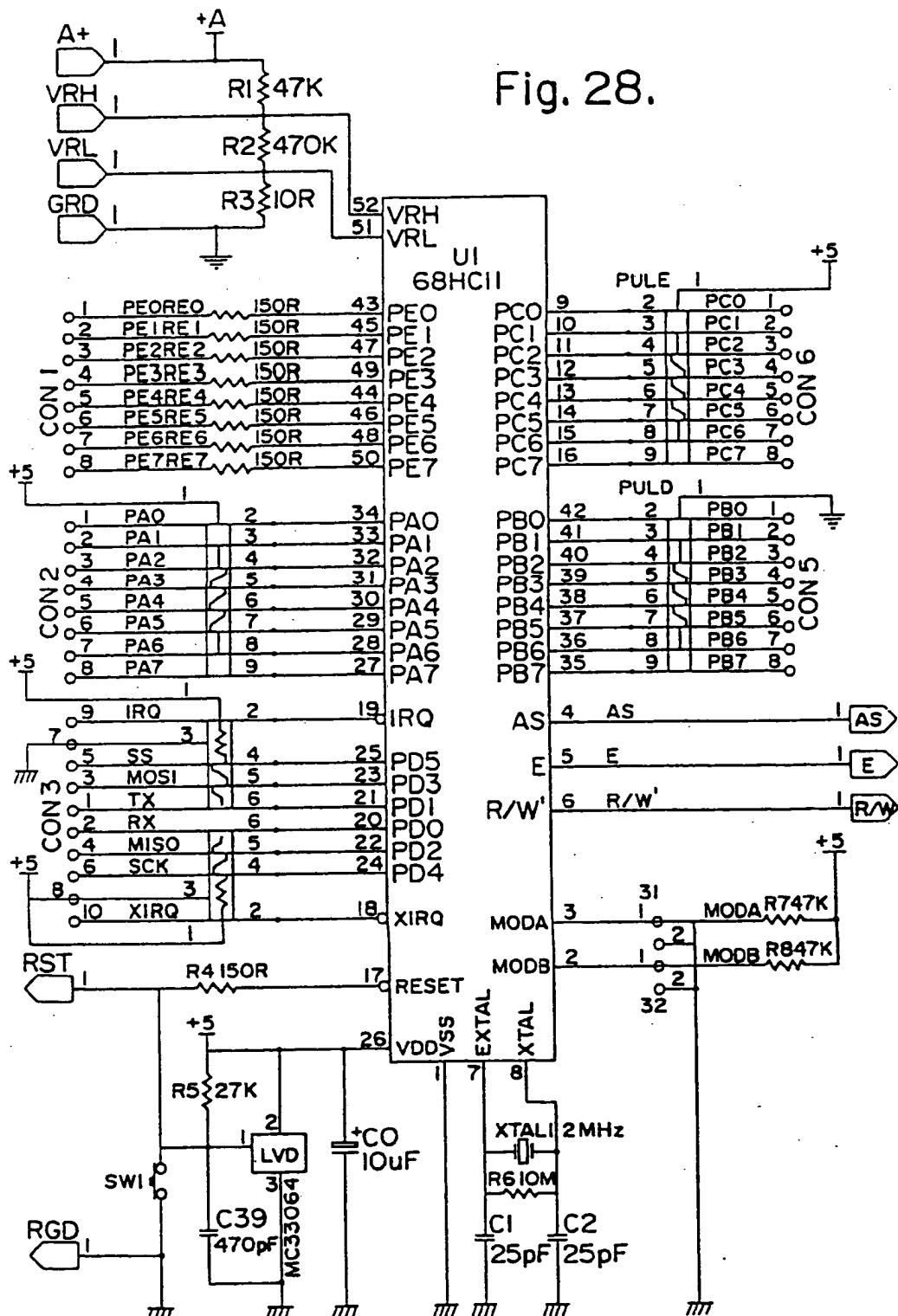
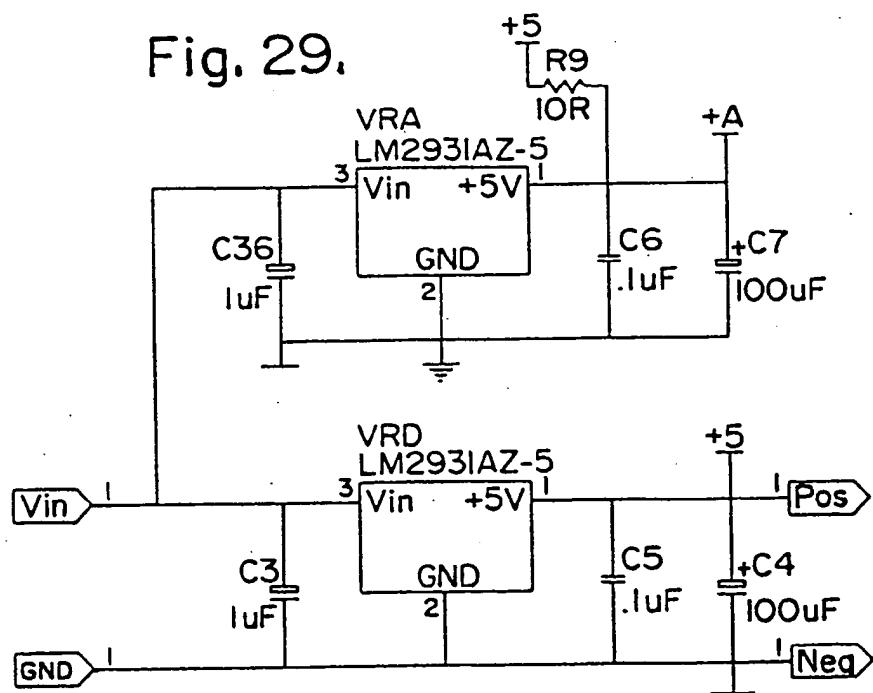


Fig. 29.



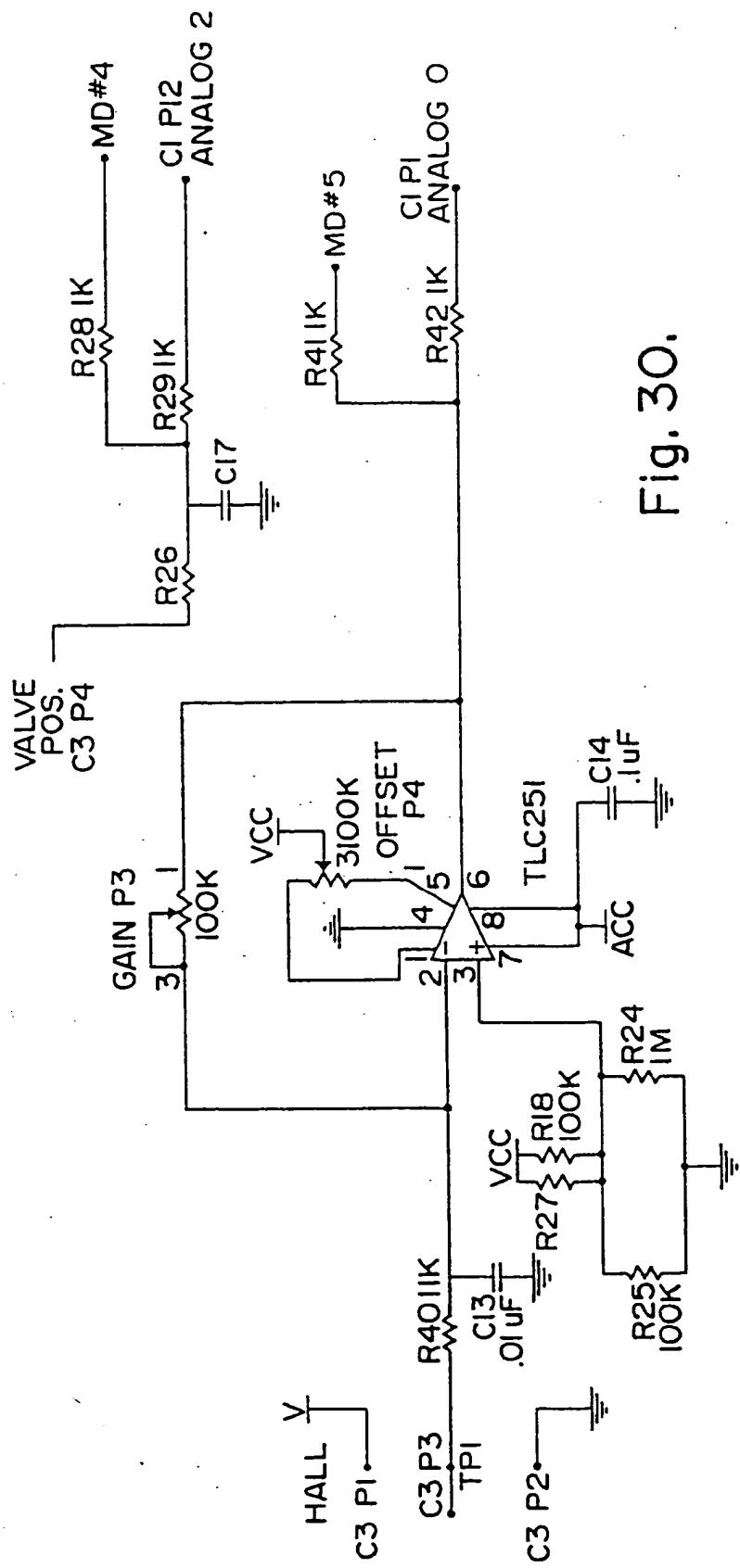


Fig. 30.

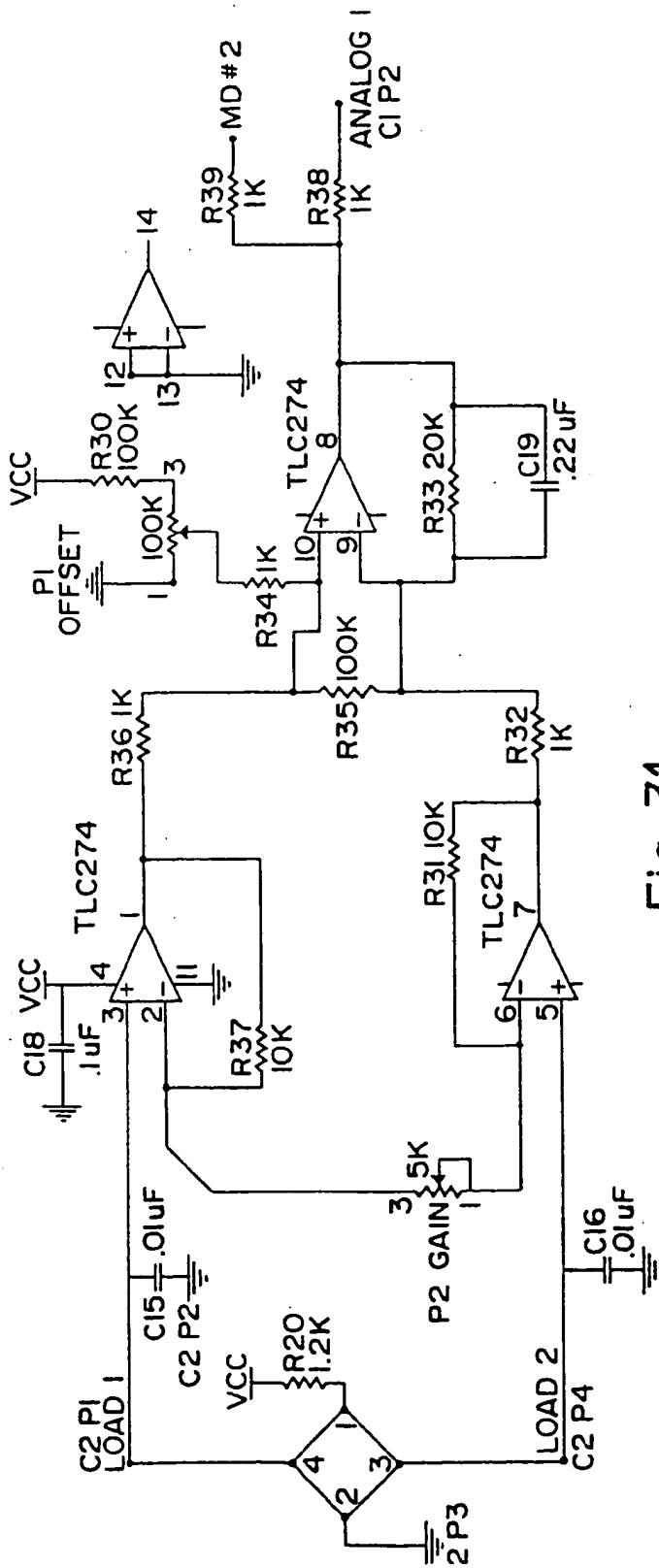


Fig. 31.

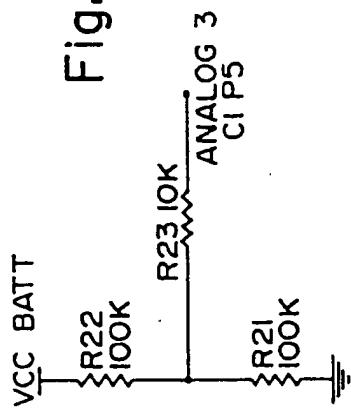
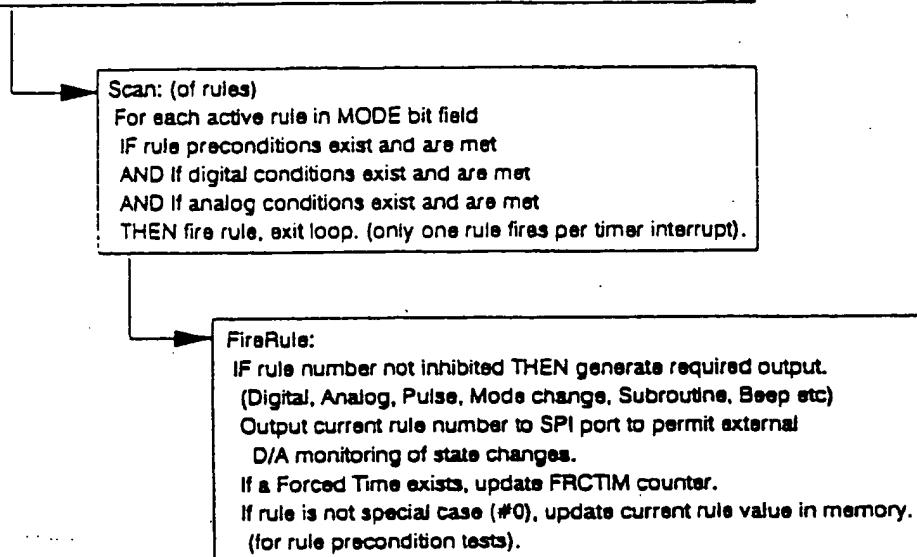


Fig. 32.

Fig. 33.

The core of the program is the Timer Interrupt Service Routine.  
Every 20 milliseconds the timer interrupts and...

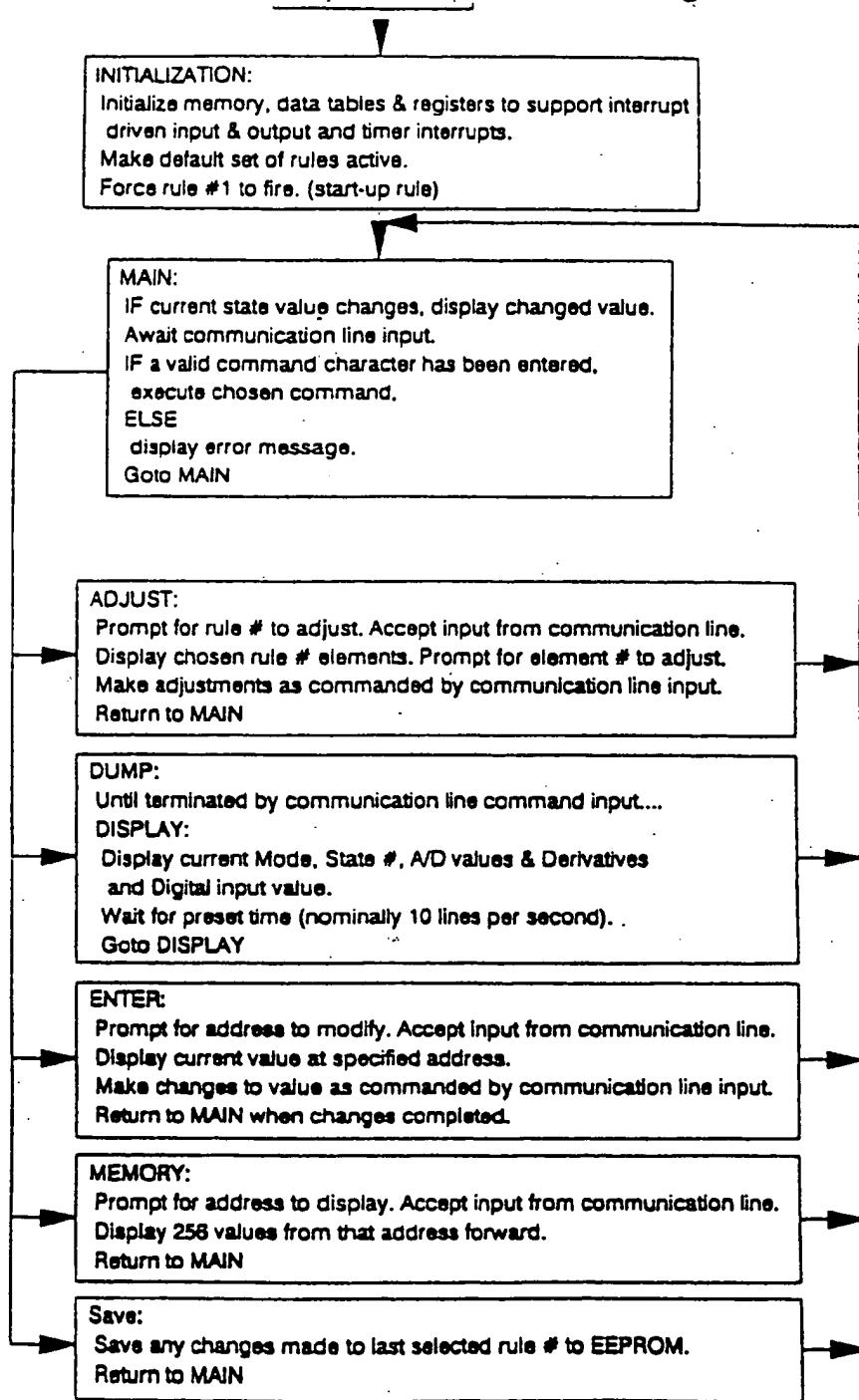
Timer Interrupt: (TimeInt @ 20 milisecond intervals)  
Get new A/D values into FIFO (first in, first out), discarding oldest values.  
Generate any required output to actuators.  
IF stop timer running, count down. On time-out, set halt flag, force  
battery cut-off rule to fire.  
IF Beep command active, execute time-outs as required, turn Beeper on/off.  
IF forced rule active, count down time. On time-out, force rule,  
goto END\_OF\_SCAN.  
IF not halted or not scanning already, scan rule table for executable rules.  
END\_OF\_SCAN: Return from timer interrupt



A standard implementation of circular buffers is used for interrupt driven  
input and output via a serial communications port on the microprocessor.  
On each serial communications interrupt...

SERIAL INTERRUPT:  
IF Receiver Interrupt THEN  
 WHILE we do not have space in the input circular buffer, WAIT.  
 Enqueue the received character, update in\_pointer & counter.  
ELSEIF  
 Transmitter interrupt THEN  
 IF we have a character to send, send it, update out\_pointer & counter.  
 ELSE  
 turn off transmitter interrupt since nothing left to send.  
 Return from SERIAL INTERRUPT

Fig. 34.



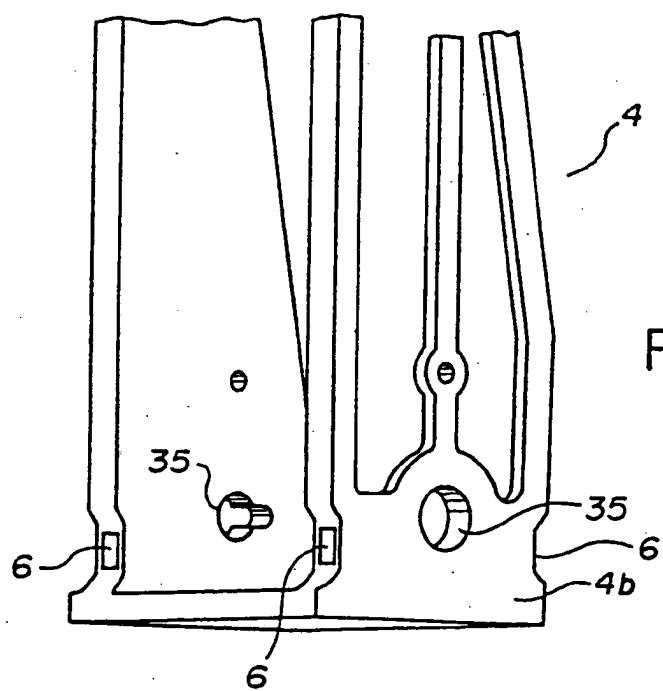


Fig. 35.



Europäisches Patentamt  
European Patent Office  
Office européen des brevets



⑪ Publication number:

0 549 855 A3

⑫

## EUROPEAN PATENT APPLICATION

⑬ Application number: 92115676.6

⑮ Int. Cl.<sup>5</sup>: A61F 2/64, A61F 2/68

⑯ Date of filing: 12.09.92

⑰ Priority: 05.12.91 US 804264

⑲ Applicant: Otto Bock Orthopädische Industrie  
Besitz- und  
Verwaltungs-Kommanditgesellschaft  
Industriestrasse  
D-37115 Duderstadt(DE)

⑳ Date of publication of application:  
07.07.93 Bulletin 93/27

㉑ Inventor: James, Kelvin B.  
7955 - 98 Avenue  
Edmonton, Alberta T6A 0B5(CA)

㉒ Designated Contracting States:

AT BE CH DE DK ES FR GB GR IE IT LI LU NL  
PT SE

㉓ Date of deferred publication of the search report:  
20.10.93 Bulletin 93/42

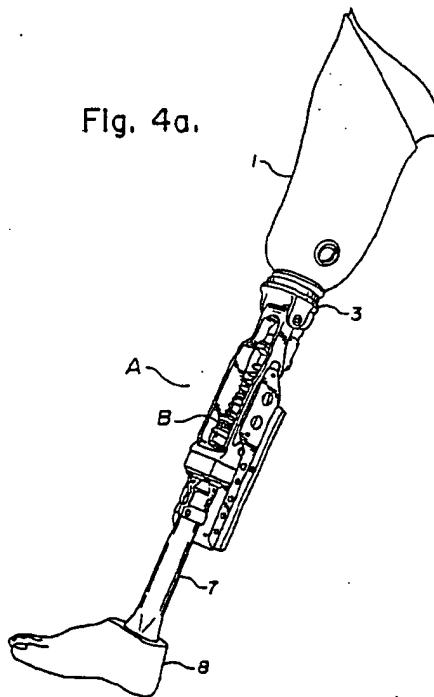
㉔ Representative: Gramm, Werner, Prof.  
Dipl.-Ing. et al  
Patentanwälte Gramm + Lins  
Theodor-Heuss-Strasse 1  
D-38122 Braunschweig (DE)

㉕ System for controlling artificial knee joint action in an above knee prosthesis.

㉖ This invention relates to an above knee prosthesis which employs a hydraulic damper to passively regulate the angular velocity or rotation of the artificial knee joint. A programmed microprocessor recognizes common gait patterns from information received from strain and knee angle sensors on the prosthesis. The microprocessor reacts at various transition points in the gait by activating a motor which in turn adjusts a valve assembly in the damper. The valve assembly is capable of variably and separately damping the knee joint motion in each of flexion and extension at the same time. Gait is improved because of the improved extent of control of knee action. In addition, distinct routines such as stair descending and sitting down can also be practised.

EP 0 549 855 A3

Fig. 4a.





European Patent  
Office

EUROPEAN SEARCH REPORT

Application Number

EP 92 11 5676

DOCUMENTS CONSIDERED TO BE RELEVANT																		
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int. Cl.5)															
D,A	FR-A-2 623 086 (ADCRO) * claims 1,2,6; figure 2 *	1,2	A 61 F 2/64 A 61 F 2/68															
A	GB-A-2 216 426 (KABUSHIKI KAISHA KOBE SEIKO SHO) * page 10, line 3 - line 21; figure 3; page 13, line 7 - page 14, line 25 *	1,2,4																
A	INTERNATIONAL JOURNAL OF ELECTRONICS vol. 64, no. 4, 1988, LONDON pages 649 - 656 CHITORE ET AL. 'DIGITAL ELECTRONIC CONTROLLER FOR ABOVE KNEE LEG PROSTHESES' * the whole document *	1,2,4																
A	GB-A- 826 314 (MAUCH) * claims 1,7; figures 1,2 *	1																
A	US-A-2 561 370 (HENSCHE ET AL.) * claim 1; figures 1,3 *	1																
			TECHNICAL FIELDS SEARCHED (Int. Cl.5)															
			A 61 F															
<p>The present search report has been drawn up for all claims</p> <table border="1"> <tr> <td>Place of search</td> <td>Date of completion of the search</td> <td>Examiner</td> </tr> <tr> <td>BERLIN</td> <td>25-03-1993</td> <td>KANAL P K</td> </tr> <tr> <td colspan="3">CATEGORY OF CITED DOCUMENTS</td> </tr> <tr> <td colspan="3">           X : particularly relevant if taken alone            Y : particularly relevant if combined with another document of the same category            A : technological background            O : non-written disclosure            P : intermediate document         </td> </tr> <tr> <td colspan="3">           T : theory or principle underlying the invention            E : earlier patent document, but published on, or after the filing date            D : document cited in the application            L : document cited for other reasons            &amp; : member of the same patent family, corresponding document         </td> </tr> </table>				Place of search	Date of completion of the search	Examiner	BERLIN	25-03-1993	KANAL P K	CATEGORY OF CITED DOCUMENTS			X : particularly relevant if taken alone Y : particularly relevant if combined with another document of the same category A : technological background O : non-written disclosure P : intermediate document			T : theory or principle underlying the invention E : earlier patent document, but published on, or after the filing date D : document cited in the application L : document cited for other reasons & : member of the same patent family, corresponding document		
Place of search	Date of completion of the search	Examiner																
BERLIN	25-03-1993	KANAL P K																
CATEGORY OF CITED DOCUMENTS																		
X : particularly relevant if taken alone Y : particularly relevant if combined with another document of the same category A : technological background O : non-written disclosure P : intermediate document																		
T : theory or principle underlying the invention E : earlier patent document, but published on, or after the filing date D : document cited in the application L : document cited for other reasons & : member of the same patent family, corresponding document																		



European Patent  
Office

EP92115676

**CLAIMS INCURRING FEES**

The present European patent application comprised at the time of filing more than ten claims.

- All claims fees have been paid within the prescribed time limit. The present European search report has been drawn up for all claims.
- Only part of the claims fees have been paid within the prescribed time limit. The present European search report has been drawn up for the first ten claims and for those claims for which claims fees have been paid.  
namely claims:
- No claims fees have been paid within the prescribed time limit. The present European search report has been drawn up for the first ten claims.

**LACK OF UNITY OF INVENTION**

The Search Division considers that the present European Patent application does not comply with the requirement of unity of invention and relates to several inventions or groups of inventions.  
namely:

1. Claims 1,2,4-12: Knee prosthesis and method for controlling the knee joint.
2. Claim 3: Hydraulic damper

- All further search fees have been paid within the fixed time limit. The present European search report has been drawn up for all claims.

- Only part of the further search fees have been paid within the fixed time limit. The present European search report has been drawn up for those parts of the European patent application which relate to the inventions in respect of which search fees have been paid.

namely claims:

- None of the further search fees has been paid within the fixed time limit. The present European search report has been drawn up for those parts of the European patent application which relate to the invention first mentioned in the claims.

1,2,4-12

namely claims: